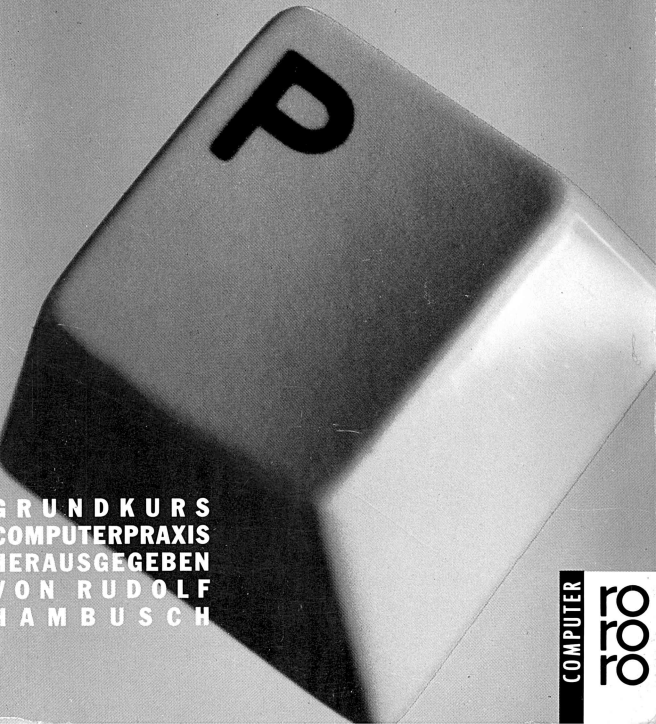
PROGRAMMIERSPRACHE TURBO-PASCAL

EINE STRUKTURIERTE EINFÜHRUNG

GREGOR KUHLMANN



GRUNDKURS COMPUTERPRAXIS
HERAUSGEGEBEN VON RUDOLF HAMBUSCH

COMPUTER
rororo

rororo computer

HERAUSGEGEBEN VON LUDWIG MOOS
UND MANFRED WAFFENDER

Die Programmiersprache Turbo-Pascal, die unter allen wichtigen Betriebssystemen von PCs läuft, hat sich in wenigen Jahren durchgesetzt. Sie wird von Programmieranfängern wie professionellen Programmierern gleichermaßen geschätzt. Ihr komfortables Textprogramm erleichtert die Eingabe und Bearbeitung von Programmen, die Menüsteuerung ermöglicht eine einfache Handhabung. Mit dieser strukturierten Einführung, die praxisnah, anschaulich und Schritt für Schritt vorgeht, läßt sich ohne große Vorkenntnisse lernen, wie Programme unter Turbo-Pascal entwickelt und zum Laufen gebracht werden.

GREGOR KUHLMANN

PROGRAMMIERSPRACHE TURBO-PASCAL

EINE STRUKTURIERTE EINFÜHRUNG

GRUNDKURS COMPUTERPRAXIS
HERAUSGEGEBEN
VON RUDOLF HAMBUSCH

ROWOHLT

131.–133. Tausend November 1999

Originalausgabe
Veröffentlicht im Rowohlt Taschenbuch Verlag GmbH,
Reinbek bei Hamburg, Dezember 1987
Copyright © 1987 by Rowohlt Taschenbuch Verlag GmbH,
Reinbek bei Hamburg
Umschlagentwurf Thomas Henning
Grafiken Ilse Parkmann, Westerkappeln
Satz Times PostScript Linotype Library, QuarkXPress 3.11
bei Langosch Grafik & DTP, Hamburg
Druck und Bindung Clausen & Bosse, Leck
Printed in Germany
ISBN 3 499 18148 7

INHALT

EDITORIAL 10

VORWORT 12

1 DAS BEDIENSYSTEM VON TURBO-PASCAL 15

1.1 Arbeitsumgebung 16
1.1.1 Turbo-Pascal starten und Beenden 16
1.1.2 Fenstertechnik 19
1.1.3 Hilfesystem 24
1.1.4 Ablauf einer Arbeitssitzung 25
1.1.5 Zusammenfassung 37
1.1.6 Übungen 38
1.1.7 Lösungen 39
1.2 Editorsystem 41
1.2.1 Überschreib- und Einfügemodus 41
1.2.2 Blockoperationen 42
1.2.3 Suchen und Ersetzen 49
1.2.4 Zusammenfassung 54
1.2.5 Übungen 55
1.2.6 Lösungen 56

2 AUFBAU EINES PASCAL-PROGRAMMS 58

2.1 Programmkopf 59
2.2 Vereinbarungsteil 61
2.3 Anweisungsteil 64
2.4 Zusammenfassung 71
2.5 Übungen 73
2.6 Lösungen 74

3 ENTWICKLUNG EINES PROGRAMMS 76

3.1 Klärung der Problemstellung: Problemanalyse 77
3.2 Datenanalyse 78
3.3 Aufbau des Lösungsweges (Algorithmus) 81
3.4 Erstellung des Programms 84
3.5 Eingabe des Programms 85
3.6 Programmtest 86
3.6.1 Syntaxtest 86
3.6.2 Logischer Test 88
3.6.3 Programm schrittweise ausführen (Trace) 90
3.6.4 Variableninhalte verändern 97
3.6.5 Laufzeitfehler 98
3.7 Dokumentation 99
3.8 Zusammenfassung 101
3.9 Übungen 102
3.10 Lösungen 103

4 AUSWAHLANWEISUNGEN 107

4.1 Einseitige Auswahl: IF – THEN 107
4.1.1 Zusammenfassung 117
4.1.2 Übungen 117
4.1.3 Lösungen 120
4.2 Zweiseitige Auswahl: IF – THEN – ELSE 121
4.2.1 Zusammenfassung 132
4.2.2 Übungen 133
4.2.3 Lösungen 135
4.3 Mehrseitige Auswahl: CASE 137
4.3.1 Zusammenfassung 145
4.3.2 Übungen 146
4.3.3 Lösungen 147

5 WIEDERHOLUNGSANWEISUNGEN 148

5.1 Fußgesteuerte Wiederholung: REPEAT – UNTIL 150
5.1.1 Zusammenfassung 154
5.1.2 Übungen 154
5.1.3 Lösungen 156

5.2 Kopfgesteuerte Wiederholung: WHILE – DO 156
5.2.1 Zusammenfassung 159
5.2.2 Übungen 160
5.2.3 Lösungen 161
5.3 Zählergesteuerte Wiederholung: FOR – TO– DO / FOR – DOWNTO – DO 161
5.3.1 Zusammenfassung 167
5.3.2 Übungen 167
5.3.3 Lösungen 168
5.4 Sprunganweisung 168
5.4.1 GOTO 168
5.4.2 BREAK 170
5.4.3 Zusammenfassung 171
5.5 Eingabeschleifen 171
5.5.1 Zusammenfassung 181
5.5.2 Übungen 182
5.5.3 Lösungen 185
5.6 Erstellen von Tabellen 186
5.6.1 Zusammenfassung 191
5.6.2 Übungen 191
5.6.3 Lösungen 194

6 ZEICHENKETTE (STRING) 196

6.1 Datentyp STRING 197
6.2 Verarbeitung von Zeichenketten 199
6.2.1 Zusammenfassung 203
6.2.2 Übungen 203
6.2.3 Lösungen 205

7 UNTERPROGRAMME 207

7.1 Benutzerdefinierte Prozeduren und Funktionen 207
7.2 Einfache, parameterlose Prozeduren 209
7.2.1 Zusammenfassung 217
7.2.2 Übungen 217
7.2.3 Lösungen 219
7.3 Prozedur mit Wertübergabe 222
7.3.1 Zusammenfassung 229

7.3.2 Übungen 230
7.3.3 Lösungen 230
7.4 Prozedur mit Variablenübergabe 232
7.4.1 Zusammenfassung 246
7.4.2 Übungen 246
7.4.3 Lösungen 248
7.5 Funktionen 250
7.5.1 Zusammenfassung 254
7.5.2 Übungen 255
7.5.3 Lösungen 257
7.6 Rekursion 259
7.6.1 Zusammenfassung 260
7.6.2 Übungen 260
7.6.3 Lösungen 261
7.7 Standardprozeduren und -funktionen 262
7.7.1 Arithmetische Funktionen 262
7.7.2 Ordinale Funktionen 265
7.7.3 Umwandlungsfunktionen 266
7.7.4 Prozeduren und Funktionen für die Bildschirmsteuerung 268
7.7.5 Prozeduren und Funktionen zur Stringverarbeitung 269
7.7.6 Sonstige Standardprozeduren und -funktionen 272

8 ÜBERSICHT ÜBER VERSCHIEDENE VERSIONEN VON TURBO-PASCAL 276

8.1 Turbo-Pascal Version 7 276
8.1.1 Das Bediensystem handhaben 276
8.1.2 Menü *Datei* 279
8.1.3 Menü *Bearbeiten* 280
8.1.4 Menü *Suchen* 281
8.1.5 Menü *Start* 282
8.1.6 Menü *Compiler* 283
8.1.7 Menü *Debug* 283
8.1.8 Menü *Tools* 287
8.1.9 Menü *Option* 288
8.1.10 Menü *Fenster* 297
8.1.11 Menü *?* 298
8.1.12 Zusammenfassung 299

8.1.13 Übungen 300
8.1.14 Lösungen 300
8.2 Turbo-Pascal Version 6 301
8.2.1 Menü *System* 301
8.2.2 Menü *File* 302
8.2.3 Menü *Edit* 303
8.2.4 Menü *Search* 304
8.2.5 Menü *Run* 305
8.2,6 Menü *Compile* 306
8.2.7 Menü *Debug* 307
8.2.8 Menü *Options* 310
8.2.9 Menü *Options/Debugger* 313
8.2.10 Menü *Options/Directories* 314
8.2.11 Menü *Options/Environment* 315
8.2.12 Menü *Window* 318
8.2.13 Menü *Help* 319
8.2.14 Zusammenfassung 320
8.2.15 Übungen 321
8.2.16 Lösungen 322
8.3 Turbo-Pascal Version 3 323
8.3.1 Aufruf 323
8.3.2 Ablauf einer Arbeitssitzung 324
8.3.3 Editierkommandos 326
8.3.4 Suchen und Ersetzen 329
8.3.5 Drucken eines Programms 330
8.3.6 Zusammenfassung 331
8.3.7 Übungen 332
8.3.8 Lösungen 333

9 ANHANG 334

9.1 Symbole für den Programmablaufplan 334
9.2 Symbole für das Struktogramm 335
9.3 ASCII-Tabelle 338
9.4 Literaturverzeichnis 339
9.5 Sachwortregister 340

EDITORIAL

Folgt man den Aussagen von Bildungs- und Wirtschaftspolitikern, so wird das Zusammenleben der Menschen in Zukunft von informationsverarbeitenden Maschinen geprägt sein: Mehr als die Hälfte aller Arbeitenden wird direkt oder indirekt mit Computern zu tun haben. Eine besondere Rolle spielt dabei der heute bereits millionenfach verbreitete Jedermanncomputer, der Personal Computer (PC). Schüler, Handwerksmeister, Rechtsanwälte, Kaufleute, Lehrer und viele andere leben, spielen und arbeiten schon heute mit diesem Gerät. Der Einsatz des persönlichen Computers ist weniger von der Fähigkeit des Benutzers geprägt, das Gerät in seiner Technizität (Hardware) zu verstehen, als vielmehr davon, es mit Hilfe der Computerprogramme (Software) zu bedienen.
Die Serie «Grundkurs Computerpraxis» erklärt Informationsverarbeitung sehr konkret und auf einfache Weise. Dabei steht das, was den Computer im eigentlichen Sinne funktionieren läßt, im Vordergrund: die Software.
Sie umfaßt

- Betriebssysteme,
- Anwenderprogramme,
- Programmiersprachen.

Ausgewählt werden Programme, die sich hunderttausendfach bewährt und einen Standard gesetzt haben, der Gefahr des Veraltens also nur in geringem Maße unterliegen.
Im «Grundkurs Computerpraxis» wird das praktische Computerwissen übersichtlich gegliedert, textverständlich strukturiert, auf das Wesentliche begrenzt und mit Grafiken, Beispielen und Übungen optimal zugänglich gemacht.

Der Herausgeber der Serie, Rudolf Hambusch, leitet am Institut für Schule und Weiterbildung des Landes Nordrhein-Westfalen unter anderem das Projekt Lehrerfortbildung in Informationsverarbeitung Wirtschaft. Die Autoren sind erfahrene Berufspädagogen, Praktiker und Mitarbeiter im Fortbildungsprojekt.

VORWORT

Die Programmiersprache Turbo-Pascal hat sich binnen weniger Jahre zu einer Standard-Sprache entwickelt. Sie wird sowohl in Schulen und Hochschulen wie auch in der professionellen Programmierpraxis eingesetzt.
Die Gründe für die starke Verbreitung von Turbo-Pascal sind unter anderem:

- das komfortable Bediensystem,
- der umfangreiche Befehlsvorrat,
- die Lauffähigkeit auf unterschiedlichen Computeranlagen und Betriebssystemen.

Das vorliegende Buch erscheint in der Reihe «Grundkurs Computerpraxis». Diese Reihe hat zum Ziel, die praktische Arbeit mit dem Personal Computer vielen zugänglich zu machen. Sie wendet sich an alle, die sich im Geschäftsleben, in der Ausbildung oder privat mit dem Computer befassen oder in Zukunft befassen wollen.

Alle Bücher dieser Reihe orientieren sich an folgenden Zielen:

- Die Darstellungen sind problembezogen.
- Die Inhalte sind auch für «Laien» leicht verständlich und nachvollziehbar.
- Alle Beispiele sollen direkt am Computer bearbeitet werden.
- Die Informationen sollen praktisch verwertbar sein.

Auf Grund dieser Zielsetzung werden in diesem Buch nicht nur einfach die Sprachelemente von Turbo-Pascal vorgestellt. Vielmehr wird an vielen Beispielen aufgezeigt, wie Programme entwickelt werden. Denn auch bei einer leistungsstarken Programmiersprache muß

die Programmentwicklung systematisch erfolgen. Entsprechend ausführlich werden die Programmbeispiele behandelt.

Das Buch ist folgendermaßen aufgebaut:

- Im Kapitel 1 wird aufgezeigt, wie unter Anwendung der Version 7 ein Turbo-Pascal-Programm in den Computer eingegeben und bearbeitet wird. Dabei werden auch die Bedienweisen von Version 6 und 5.5 berücksichtigt. Nähere Erläuterungen zu einzelnen Versionen finden Sie im Kapitel 8.
- Kapitel 2 beschreibt den grundsätzlichen Aufbau eines Pascal-Programms.
- In den Kapiteln 4 und 5 wird dargelegt, wie Programmabläufe durch Auswahl- und Wiederholungsanweisungen gesteuert werden.
- Kapitel 6 behandelt die Verarbeitung von Zeichenketten.
- Im Kapitel 7 werden die unterschiedlichen Formen von Unterprogrammen beschrieben.
- Am Ende eines jeden Kapitels finden Sie eine zusammenfassende Übersicht als Merkhilfe sowie zur Vertiefung des Gelernten einen Übungsteil mit Lösungen.

Auf Grund dieses Ansatzes können in diesem Buch nicht alle Sprachelemente von Turbo-Pascal dargelegt werden. Im Band 2 (rororo computer Nr. 8155) erfolgt die Behandlung der Window-Technik, der Datenstrukturen Record, Datei (sequentiell und indexsequentiell) sowie Zeiger und verkettete Liste. Im Band Turbo-Pascal für Windows werden die Programmierung von Grafiken sowie die objektorientierte Programmierung unter Windows thematisiert.
Die Firma Borland hat in den vergangenen Jahren das Turbo-Pascal-System weiterentwickelt. Die verschiedenen Versionen von Turbo-Pascal unterscheiden sich, bezogen auf eine strukturierte Einführung, hauptsächlich in der Bedienweise.
Hinsichtlich der Pascal-Sprachelemente bestehen kaum Unterschiede. In den Programmen werden sie durch Kommentare, zum Beispiel

```
uses CRT; (*ab Version 4 *).
```

gekennzeichnet.
Die Ausführungen dieses Buches gelten somit für alle Versionen von Turbo-Pascal.

Ich danke Herrn Dipl.-Handelslehrer Gerhard Haase sowie Herrn StD Joachim Röhl für Beratung und Korrektur bei der Erstellung des Manuskripts.

Gregor Kuhlmann

Zur Erleichterung der Arbeit mit diesem Buch können die von den Autoren verwendeten Beispiele auf einer Diskette gesondert bestellt werden. **Bestellnummer:** D3-8148 (3,5"-Diskette) für *DM 22,00.* Der Preis ist inklusive Porto und Verpackung. Die Bestellung erfolgt durch Zusendung eines *Verrechnungsschecks (*bei Auslandsaufträgen eines auf DM lautenden *Euroschecks)* oder durch Überweisung auf das *Konto 4001186* bei der *Stadtsparkasse Lengerich, BLZ 40154476* **(Adresse bitte im Feld Verwendungszweck mit angeben!)** an:
DISKSERVICE Hennig, Postfach 1574, D-49525 Lengerich

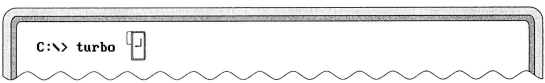
1 DAS BEDIENSYSTEM VON TURBO-PASCAL

Ein Computer ist eine «dumme» Maschine. Gleichgültig, ob es sich um einen Großcomputer oder um einen Personal Computer (PC) handelt, in jedem Fall muß ihm zunächst ein Programm eingegeben werden, damit er arbeiten kann.
Ein Programm besteht aus Anweisungen an einen Computer, die in einer Sprache geschrieben sind, die ein Computer «verstehen» kann. Eine solche Sprache nennt man Programmiersprache. Es gibt über hundert Programmiersprachen: COBOL, FORTRAN, BASIC, Pascal, LISP... In diesem Buch wird die Programmiersprache Pascal unter besonderer Berücksichtigung von Turbo-Pascal erläutert.
Programme müssen in den Computer eingegeben werden. Die Eingabe wird über die Tastatur des Computers vorgenommen. Damit Sie als Programmierer es einfach haben, Anweisungstexte einzugeben oder fertige Programme «laufen» zu lassen, verfügt Turbo-Pascal über ein komfortables Bediensystem.
Das leistungsfähige Editiersystem, ausgestattet mit einer nutzerfreundlichen Fenstertechnik und Mausunterstützung, erleichtert die Programmeingabe. Auch bietet das System Funktionen an, die ein schnelles Auffinden von Programmierfehlern ermöglichen. Um die Leistungen der integrierten Entwicklungsumgebung nutzen zu können, muß ihre Handhabung bekannt sein. Deshalb wird, ausgehend von praktischen Beispielen und Problemstellungen, zunächst die Arbeitsumgebung, das Editor- und Debuggersystem behandelt. Eine systematische Darstellung der Bediensysteme der Versionen 3, 6 und 7 finden Sie im Kapitel 8.

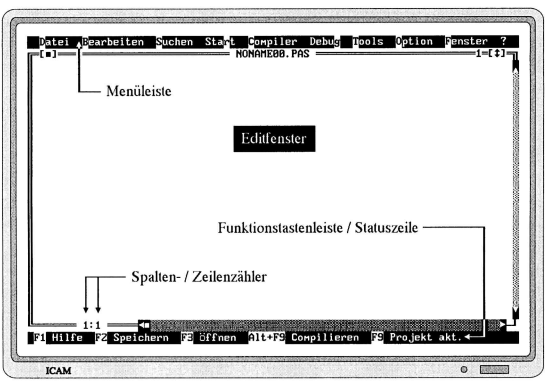
1.1 Arbeitsumgebung

1.1.1 Turbo-Pascal starten und beenden

Starten
Das Turbo-Pascal-System starten Sie durch Eingabe von



Das System meldet sich mit nachstehendem Eingangsbildschirm:



Der Bildschirm teilt sich auf in drei Bereiche: Menüleiste, Editfenster und Funktionstastenleiste.

Menüleiste
Über die Menüleiste werden Pull-Down-Menüs (Rolladenmenüs) geöffnet, in denen Kommandos angewählt werden. Für den Zugriff auf die Menüleiste stehen Ihnen mehrere Wege offen.

1. Möglichkeit:
Verwenden Sie eine Maus, so stellen Sie den Mauszeiger auf den gewünschten Menüpunkt und klicken die linke Taste kurz an.

2. Möglichkeit:
Betätigen Sie die Taste F10, und geben Sie anschließend den hervorgehobenen Buchstaben des Menüpunkts ein, den sie anwählen wollen.

3. Möglichkeit:
Aktivieren Sie mittels F10 die Menüleiste. Stellen Sie den Cursorbalken mit Hilfe der Tasten ← bzw. → auf den gewünschten Menüpunkt, und betätigen Sie die Taste ↵.

4. Möglichkeit:
Während Sie die Alt-Taste niedergedrückt halten, schlagen Sie den hervorgehobenen Buchstaben des Menüpunkts an.
Das System reagiert mit der Anzeige des Untermenüs. Wenn Sie beispielsweise die Tastenkombination Alt D oder die Tastenfolge F10 D eingeben oder mit der Maus den Menüpunkt *Datei* anklicken, baut das System folgendes Menü auf:

In dem Pull-Down-Menü werden die zugehörigen Funktionen angezeigt. Drei Punkte hinter einem Funktionsnamen weisen darauf hin, daß noch ein Unterfenster folgt. Um eine Funktion aufzurufen, stehen Ihnen mehrere Möglichkeiten zur Verfügung:

1. Möglichkeit:
Sie stellen den Cursorbalken mit Hilfe der Pfeiltasten auf den Namen der gewünschten Funktion und bestätigen Ihre Wahl mit [↵].

2. Möglichkeit:
Sie geben den Buchstaben ein, der im Funktionsnamen hervorgehoben ist.

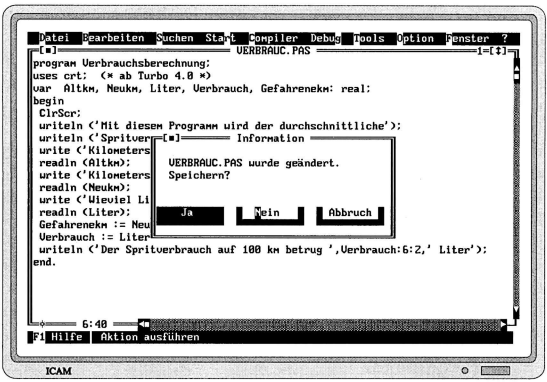
3. Möglichkeit:
Sie klicken mit der Maus den Namen der Funktion an.

> Durch Betätigen von [Esc] heben Sie die Anwahl des Menüpunkts wieder auf.

Editfenster
Im Editfenster erfolgt die Eingabe und die Bearbeitung des Programmtextes.

Funktionstastenleiste/Statuszeile
Die Funktionstastenleiste ermöglicht den direkten Aufruf von häufig benötigten Kommandos. Wird ein Menüpunkt aufgerufen, so ändert sich der Inhalt der Funktionstastenleiste. Es wird jedoch stets die Hilfetaste [F1] angeboten sowie eine Statuszeile angezeigt, in der Hinweise auf den aktuellen Menüpunkt bzw. auf eine gerade laufende Aktion gegeben werden.

Beenden
Die Arbeit mit Turbo-Pascal wird mittels der Tastenkombination [Alt] [X] beendet. Dazu wird die [Alt]-Taste niedergedrückt und die Taste [X] angeschlagen. Anschließend werden beide Tasten wieder losgelassen. Falls das Programm im Editfenster seit der letzten Änderung noch nicht gespeichert worden ist, macht das Pascal-System Sie darauf aufmerksam, wie aus der nachstehenden Abbildung zu entnehmen ist.



In dem eingeblendeten Fenster werden neben dem Text auch sogenannte **Aktionsschalter**, wie z. B. *Ja*, angezeigt. Aktionsschalter erkennen Sie an der angedeuteten dreidimensionalen Darstellung. Über diese Schalter werden bestimmte Aktionen ausgelöst. Sie betätigen einen Aktionsschalter, indem Sie ihn

(a) mit der Maus anklicken oder

(b) mittels der Tastenkombination Alt und hervorgehobener Buchstabe ansprechen oder

(c) mittels der ⇆-Taste aktivieren. ↵ löst die Aktion aus.

Während die Aktionsschalter *Ja* und *Nein* die Arbeitssitzung beenden, führt der Schalter *Abbruch* zurück in den vorherigen Arbeitszustand. *Abbruch* bewirkt stets die Aufhebung einer eingeleiteten Aktion.

1.1.2 Fenstertechnik

Fensterstruktur im Überblick

Turbo-Pascal wendet konsequent die Fenstertechnik an. Sie können sich eine Vielzahl von Fenstern auf dem Bildschirm aufbauen, die sich ganz oder teilweise überlagern. Das versetzt Sie in die Lage, sich eine optimale Arbeitsoberfläche zu schaffen. Allerdings kann im

Anfang die Vielfalt der Fenster verwirren. Deshalb sei hier ein kurzer Überblick über die Fensterarten und ihre Funktionen gegeben. Beim Programmstart bietet das System das **Editfenster** an. In diesem Fenster nehmen Sie die Eingabe bzw. Bearbeitung von Programmtexten (Quellprogrammen) vor. In dieses Fenster können Sie das **Überwachungs-** bzw. **Watchfenster** einblenden. Das Überwachungsfenster ermöglicht es Ihnen, sich den Inhalt von ausgewählten Speicherplätzen bzw. Variablen anzeigen zu lassen. Somit können Sie beim Programmtest leicht die Wertentwicklung in den Variablen verfolgen. Für die Datenausgabe verwendet Turbo-Pascal den sogenannten **Ausgabebildschirm**. Er nimmt stets den gesamten Bildschirm in Anspruch. Möchten Sie die Datenausgabe sowie auch das Quellprogramm gleichzeitig auf dem Bildschirm beobachten, so blenden Sie das sogenannte **Ausgabefenster** ein. Das Ausgabefenster stellt einen Ausschnitt des Ausgabebildschirms dar.

Fenster installieren

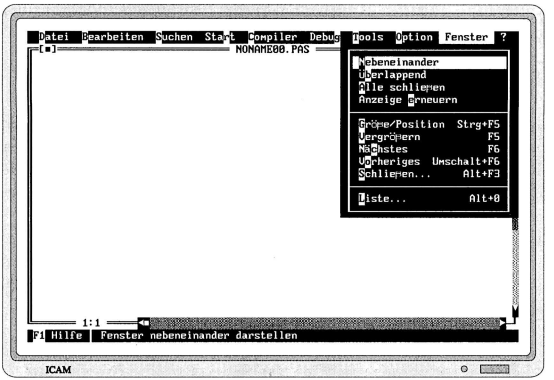
Das Editfenster wird unmittelbar vom System vorgegeben. Die übrigen Fenster rufen Sie über einen Menüpunkt auf. Der Weg zum Aufruf eines Menüpunktes wird kurz und präzis wie folgt angegeben:

Hauptmenüpunkt/Auswahlpunkt/...

Die oben erwähnten Fenster werden in den verschiedenen Versionen von Turbo-Pascal wie folgt aufgerufen:

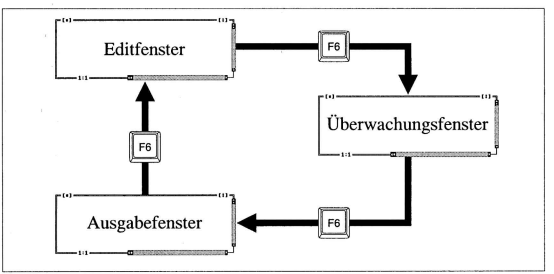
Fenstertyp	Version 7	Version 6	Version 5.x
Überwachungsfenster	Debug/ Überwachte Ausdrücke	Window/Watch	bereits vorgegeben
Ausgabebildschirm	Debug/ Ausgabebildschirm	Window/ UserScreen	Run/ UserScreen
Ausgabefenster	Debug/ Ausgabefenster	Window/Output	–

In dem nun angezeigten Menü wählen Sie den gewünschten Fenstertyp.



Fenster aktivieren

Das System numeriert die Fenster. Ein bestimmtes Fenster sprechen Sie durch Eingabe der Tastenkombination Alt und Fensternummer an. Welches Fenster gerade aktiv ist, erkennen Sie an der doppelten Umrandung. Wollen Sie beispielweise aus dem Fenster Nr. 3 in das Editfenster mit der Nr. 2 springen, so geben Sie Alt 2 ein.
Um das Fenster zu wechseln, können Sie sich auch der Funktionstaste F6 bzw. ⇧ F6 bedienen.

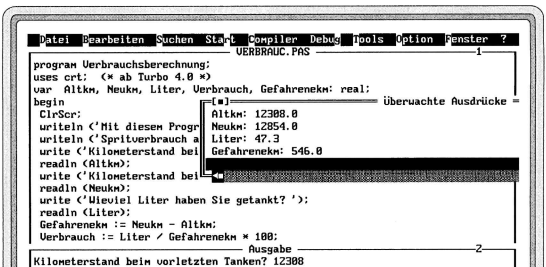


Bei Einsatz einer Maus aktivieren Sie ein Fenster, indem Sie an einer beliebigen Stelle den Rand des Fensters anklicken.

Fenster verschieben

Sie können die Fenster beliebig auf dem Bildschirm anordnen. Setzen Sie eine Maus ein, so fahren Sie den Mauszeiger auf den Rahmen des betreffenden Fensters und drücken die linke Maustaste. Der Rahmen verändert sich. Durch Verschieben der Maus plazieren Sie das Fenster an eine andere Stelle im Bildschirm. Haben Sie die gewünschte Stelle erreicht, lassen Sie die Maustaste los.

Falls Ihnen keine Maus zur Verfügung steht, aktivieren Sie das betreffende Fenster und betätigen Strg F5. Mittels der Pfeiltasten verschieben Sie nun das Fenster an eine andere Stelle im Bildschirm. Durch ↵ wird das Fenster an der aktuellen Position fixiert.

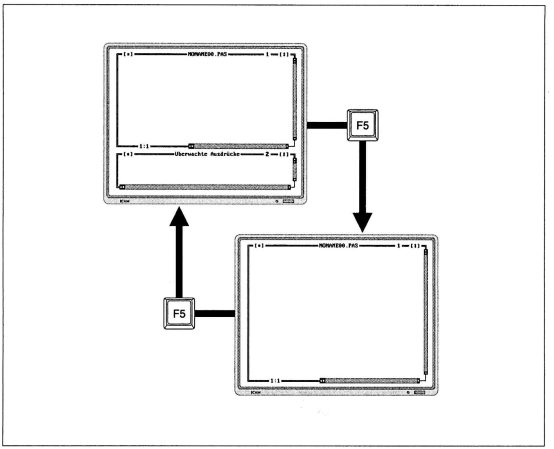


Zoomen/Vergrößern/Verkleinern

Damit die Fenster in einer für die Arbeitssituation erforderlichen Größe auf dem Bildschirm angezeigt werden, können Sie ihre Größe ändern.

Mittels F5 wird die **Zoom**-Funktion aufgerufen. Der Zoom-Schalter stellt ein aktives Fenster, das bislang nur einen Teil des Bild-

schirms bedeckte, auf die volle Bildschirmgröße um. Bei erneutem Aufruf der Zoom-Funktion wird ein Fenster, das in maximaler Größe dargestellt ist, in die vorherige Größe zurückgesetzt.



Diese Wirkung erzielen Sie bei Einsatz einer Maus dadurch, daß Sie das Pfeilsymbol [↑] im aktiven Fenster anklicken. Um das Fenster wieder auf die ursprüngliche Größe zurückzuführen, klicken Sie den Doppelpfeil an.

Zur Vergrößerung bzw. Verringerung der Höhe und Breite eines Fensters fahren Sie den Mauszeiger auf die rechte untere Ecke des Fensters und drücken die linke Maustaste. Durch Verschieben der Maus ändern Sie die Maße und die Lage des Fensters. Haben Sie die gewünschte Lage oder Größe eingestellt, lassen Sie die Maustaste los.

Steht Ihnen keine Maus zur Verfügung, so schalten Sie zunächst mit Strg F5 die *Fenster/Größe/Position*-Funktion ein. Mittels der Tastenkombination ◇ und Pfeiltasten bewirken Sie die Größen- und Lageveränderung des Fensters.

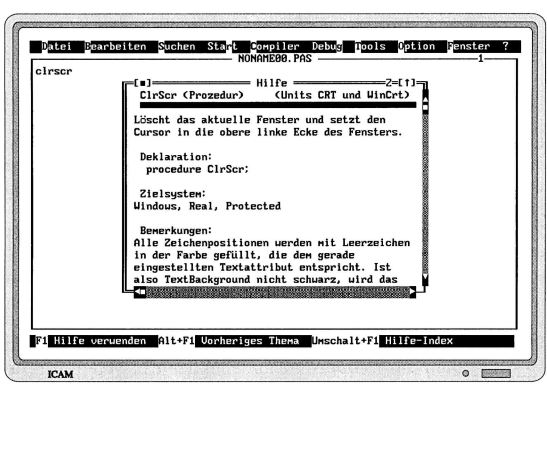
Fenster aufheben

Um ein Edit-, Überwachungs- bzw. Ausgabefenster aufzuheben, aktivieren Sie das Fenster und betätigen die Tastenkombination Alt F3. Bei Einsatz einer Maus klicken Sie das Schließsymbol [■] in der linken oberen Fensterecke an.

1.1.3 Hilfesystem

Turbo-Pascal Version 7 enthält ein umfassendes Hilfesystem. Mit F1, Strg F1 oder ⇧ F1 fordern Sie Texte an, die Begriffe oder Funktionen des Sprach- und Entwicklungssystems näher erläutern.

Beispiel:
Bei der Programmeingabe haben Sie das Befehlswort ClrScr geschrieben und möchten sich ohne Blick ins Handbuch informieren, wie Konstanten zu vereinbaren sind. Stellen Sie dazu den Cursor auf das Befehlswort ClrScr. Mit der Tastenkombination Strg F1 rufen Sie folgendes Hilfefenster auf:



Ist der Hilfetext länger als das Fenster, so können Sie ihn mit Hilfe von [↑] bzw. [↓] nach oben oder unten «rollen» lassen. Beim Mauseinsatz wird der «Rollbalken», der das Fenster rechts begrenzt, zum Verschieben des Textes verwendet. Positionieren Sie dazu den Mauszeiger auf das obere oder untere Pfeilsymbol, und halten Sie die Maustaste so lange nieder, bis der gewünschte Textausschnitt im Fenster steht. Befindet sich die Marke im Rollbalken oben oder unten, haben Sie den Textanfang bzw. das Textende erreicht.
In den Hilfefenstern sind einige Begriffe farblich hervorgehoben. Es handelt sich hier um Begriffe, zu denen Sie noch weitere Informationen abrufen können. Stellen Sie dazu den Cursor mittels der Pfeiltasten oder [⇆] auf den betreffenden Begriff, oder klicken Sie ihn mit der Maus an. Zum Abruf der zugehörigen Informationen schlagen Sie die Taste [↵] an, oder Sie betätigen zweimal schnell nacheinander die Maustaste (Doppelklick).
Die Tastenkombination [Alt] [F1] führt Sie zum vorhergehenden Fenster zurück. Durch [Esc] verlassen Sie das Hilfesystem und kehren zum Ausgangspunkt Ihrer Hilfeanforderung zurück.
Um zu einem Kommando oder Schlüsselwort nähere Erläuterungen zu erhalten, können Sie auch den Weg über den Hilfeindex nehmen. Sie rufen das Stichwortverzeichnis mit [⇧] [F1] auf. Tippen Sie nun beispielsweise den Begriff ClrScr ein, so wird der Cursorbalken auf den gewünschten Begriff gestellt. Durch [↵] oder Doppelklick wird der Hilfetext eingeblendet. Das Hilfesystem steht in jeder Phase einer Arbeitssitzung zur Verfügung. Der Programmierer kann sich somit rasch die benötigten Informationen auf den Bildschirm holen.

1.1.4 Ablauf einer Arbeitssitzung

Nach dem Start von Turbo-Pascal befinden Sie sich im Editfenster. Es empfiehlt sich, vor der Eingabe des Programms ein eigenes Unterverzeichnis festzulegen, in dem Sie das Programm später speichern wollen. Ohne Vereinbarung eines eigenen Unterverzeichnisses werden Ihre Programme in das gerade aktuelle Verzeichnis geschrieben. Sie laufen dann Gefahr, daß Sie die Übersicht verlieren.

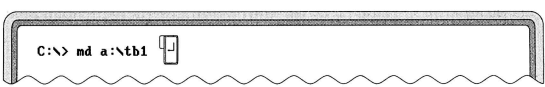
Unterverzeichnis einrichten
Falls Sie ein Unterverzeichnis einrichten wollen, brauchen Sie trotz des laufenden Pascal-Systems die Arbeitssitzung nicht zu beenden.

Sie wechseln wie folgt auf die Betriebssystemebene und können dann Betriebssystemkommandos ausführen lassen.

Aufruf des Betriebssystems:

Version 7	Version 6	Version 5.x
Datei/	File/	File/
DOS aufrufen	DOS shell	OS shell

Beispiel:
Sie möchten für Ihre Pascal-Programme auf der Diskette im Laufwerk a: das Unterverzeichnis TB1 einrichten. Das Betriebssystemkommando lautet dann:

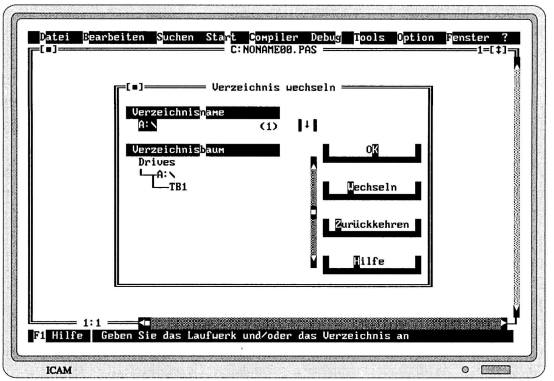


Nähere Angaben zu dem Kommando md (**m**ake **d**irectory = lege Verzeichnis an) und zur Bildung von Verzeichnisnamen finden Sie bei: Freese, Peter: Standardbetriebssystem MSDOS, rororo 8145.
Durch Eingabe von **exit** ↵ kehren Sie in das Pascal-System zurück.

Unterverzeichnis festlegen
Ein Unterverzeichnis wird auf folgende Weise festgelegt:

Version 7	Version 6	Version 5.x
Datei/	File/	File/
Verzeichnis wechseln	Change Dir	Change dir

Das Dialogfenster zu *Verzeichnis wechseln* verfügt über Aktionsschalter und ein Eingabefeld (1).



Zur Angabe des Verzeichnisnamens stehen Ihnen mehrere Möglichkeiten offen.

1. Möglichkeit:
Sie schreiben den Namen des gewünschten Verzeichnisses unmittelbar in die Verzeichniszeile (1), zum Beispiel a:\tb1, Abschluß der Eingabe mit ↵ oder durch Betätigen des Aktionsschalters *OK*.
Auch wenn der Verzeichnisname länger ist als der Schreibraum, der in dem Eingabefeld zur Verfügung steht, so können Sie weiterschreiben. Die Eingabezeile wird automatisch weitergeschoben.
In der Regel schlägt das System das Verzeichnis, aus dem Turbo Pascal gestartet wurde, als aktuelles Verzeichnis vor. Sobald Sie eine Zeichen- oder Zifferntaste anschlagen, verschwindet der Vorschlag. Wenn Sie ihn übernehmen wollen, bestätigen Sie ihn. Falls Sie ihn aber verändern oder ergänzen wollen, betätigen Sie die Taste → oder die Taste Ende, nicht aber die Taste ↓. Anschließend können Sie den Vorschlag bearbeiten.

2. Möglichkeit:
Sie wählen das Laufwerk und das Verzeichnis in dem Verzeichnisbaumfenster aus. Steht Ihnen keine Maus zur Verfügung, stellen Sie den Cursor mittels [⇆] in das Verzeichnisbaumfenster. Wollen Sie das Laufwerk wechseln, so positionieren Sie den Cursorbalken auf *Drives* und betätigen die Taste [↵]. Damit rufen Sie die Funktion des Schalters *Wechseln* auf, die die verfügbaren Laufwerke anzeigt. Verschieben Sie nun den Cursorbalken auf das gewünschte Laufwerk und drücken erneut die Taste [↵], so wird der Verzeichnisbaum der Platte bzw. Diskette angezeigt. Stellen Sie nun den Cursorbalken auf das gewünschte Verzeichnis, und betätigen Sie mit der Tastenkombination [Alt] [K] den *OK*-Schalter.
Bei Mauseinsatz klicken Sie zur Anwahl des Laufwerks das Feld *Drives* an und führen einen Doppelklick aus. In der Liste der verfügbaren Laufwerke klicken Sie das gewünschte Laufwerk an und vollziehen anschließend einen Doppelklick. Das System reagiert mit der Anzeige des Verzeichnisses. Sie klicken das gewünschte Verzeichnis an und bestätigen die Wahl mit einem Doppelklick.

3. Möglichkeit:
Ein Wechsel in ein Verzeichnis, in dem Sie während der Arbeitssitzung schon einmal gearbeitet haben, ist schnell vollzogen. Falls der Cursor nicht in der Eingabezeile (1) steht, bewegen Sie ihn mit [Alt] [N] dorthin und betätigen die Taste [↓]. In einem weiteren Fenster wird Ihnen die Liste bereits früher eingegebener bzw. angewählter Verzeichnisse angezeigt. Diese Liste wird **Eingabeaufzeichnungsliste** genannt. Stellen Sie den Cursor auf den Namen des gewünschten Verzeichnisses, und bestätigen Sie die Wahl mit [↵] oder mit einem Doppelklick. Der Verzeichnisname wird in die Eingabezeile übernommen. Durch Betätigen des Aktionsschalters *OK* wird der Wechsel durchgeführt.
Die Eingabeaufzeichnungsliste erfüllt eine Art Protokollfunktion. Sie kann stets im Zusammenhang mit Eingabefeldern mittels [↓] oder durch Anklicken des Listensymbols ↓ aufgerufen werden.
Möchten Sie an den Ausgangspunkt Ihrer Suche zurückkehren, so bedienen Sie den Aktionsschalter *Zurückkehren*.
Verwenden Sie eine Maus, so gestaltet sich der Ablauf des Verzeichniswechsels ein wenig einfacher, weil Sie jeweils nur die entsprechenden Schalter oder Felder anzuklicken brauchen.

Eingabe des Programms

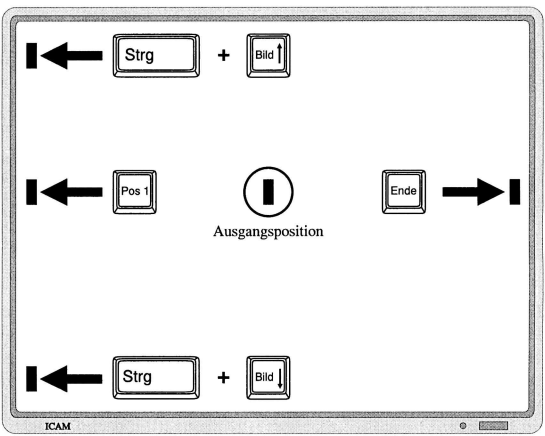
Nach Festlegung des Unterverzeichnisses können Sie nun mit der Eingabe des Programmtextes beginnen. Geben Sie beispielsweise folgendes Programm ein:

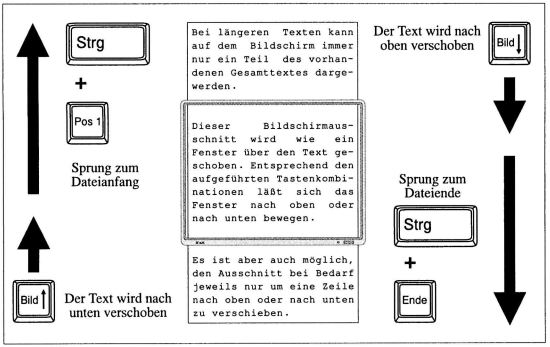
```
program Verbrauchsberechnung;
uses crt;                                          (* ab Version 4 *)
var  Altkm, Neukm, Liter, Verbrauch, Gefahrenekm: real;
begin
  ClrScr;
  writeln ('Mit diesem Programm wird der durchschnittliche');
  writeln ('Benzinverbrauch auf 100 km berechnet');
  write ('Kilometerstand beim vorletzten Tanken? ');
  readln (Altkm);
  write ('Kilometerstand beim letzten Tanken? ');
  readln (Neukm);
  write ('Wieviel Liter haben Sie getankt? ');
  readln (Liter);
  Gefahrenekm := Neukm - Altkm;
  Verbrauch := Liter / Gefahrenekm * 100;
  writeln ('Der Benzinverbrauch auf 100 km betrug ',
           Verbrauch:8:2,' Liter');
end.
```

Die Erläuterungen zu dem Programm finden Sie im Kapitel 2. Die Handhabung des Editors wird im Kapitel 1.2 näher beschrieben. Hier mögen einige Hinweise genügen, die für das unmittelbare Korrigieren von Tippfehlern und für die Ansteuerung von Textstellen wichtig sind.

Taste	Bedeutung
←	Zeichen links vom Cursor löschen
Entf	Zeichen auf Cursorposition löschen
Ende	Cursor an das Zeilenende stellen
Posl	Cursor an den Zeilenanfang stellen
Strg Posl	Cursor an den oberen Bildrand stellen
Strg Ende	Cursor an den unteren Bildrand stellen
Bild ↓	Text um eine Bildschirmseite nach oben schieben
Bild ↑	Text um eine Bildschirmseite nach unten schieben
Strg Bild ↑	Sprung zum Textanfang
Strg Bild ↓	Sprung zum Textende

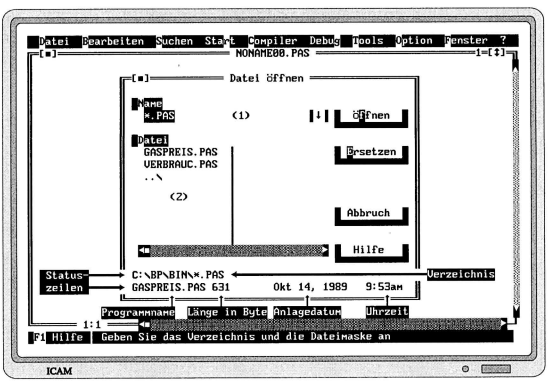
Falls Sie eine Maus einsetzen, veranlassen Sie das Rollen des Textes über die Pfeil-oben- bzw. Pfeil-unten-Symbole am Rollbalken des Editfensters. Um den Text nach rechts oder links zu verschieben, benutzen Sie das Pfeil-links- bzw. Pfeil-rechts-Symbol am unteren Fensterrand.





Laden eines Programms

Der Ladevorgang wird mittels F3 eingeleitet. Dadurch wird ein Fenster geöffnet, in dem der Name des gewünschten Programms einzugeben ist:

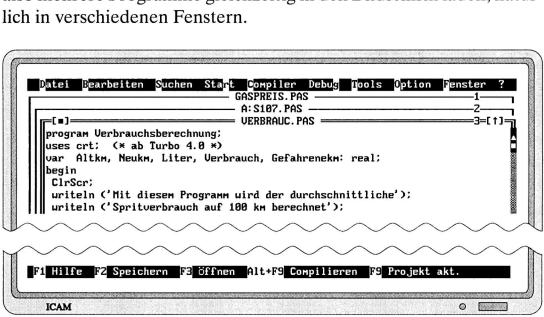


Zur Namensangabe stehen Ihnen mehrere Möglichkeiten offen.

1. Möglichkeit:
Sie schreiben den Namen des Programms, das Sie laden wollen, direkt in die Namenszeile (1) und schließen die Eingabe mit ↵ ab oder betätigen den Aktionsschalter *Öffnen*.

2. Möglichkeit:
Sie stellen den Cursor mittels ⇆ oder per Maussteuerung in die Liste der Dateinamen (2). Dort stellen Sie den Cursorbalken auf den Namen des gewünschten Programms und bestätigen Ihre Wahl mittels ↵ oder Doppelklick.

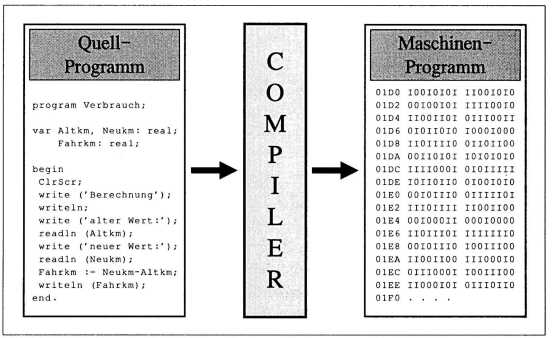
3. Möglichkeit:
Falls Sie während der Arbeitssitzung bereits mehrere Programme angesprochen haben, können Sie über die Eingabeaufzeichnungsliste auf sie zugreifen. Diese Liste rufen Sie durch Aktivieren des Symbols ↓ auf. Zur Wahl des gewünschten Programms stellen Sie den Cursorbalken auf den betreffenden Namen. Mit ↵ oder mit einem Doppelklick bestätigen Sie die Wahl. Der Aktionsschalter *Öffnen* setzt den Ladevorgang in Gang.
Steht bereits ein Programm im Editfenster, so wird für das nun einzulesende Programm ein weiteres Editfenster geöffnet. Sie können also mehrere Programme gleichzeitig in den Bildschirm laden, natürlich in verschiedenen Fenstern.



Falls Sie das zu ladende Programm nicht in einem neuen Fenster, sondern in dem gerade aktiven Editfenster bearbeiten wollen, betätigen Sie den Aktionsschalter *Ersetzen*. Er bewirkt, daß das alte Programm im aktiven Editfenster überschrieben wird.

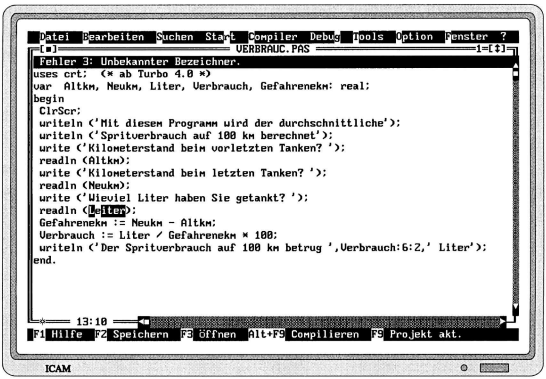
Übersetzen des Quellprogramms mit Syntaxprüfung

Wenn Sie die Eingabe Ihres Programms (Quellprogramm) abgeschlossen haben, muß es in eine vom Computer ausführbare Form gebracht werden. Das Quellprogramm ist in das Maschinenprogramm zu übersetzen. Diese Aufgabe übernimmt der **Compiler**.



Der Übersetzungsvorgang kann über Compiler/Compilieren aufgerufen werden. Unter Verwendung der Funktionstasten starten Sie den Compiler mittels F9 oder Alt F9 oder Alt C C. Ist das Programm hinsichtlich der Schreibregeln (Syntax) in Ordnung, so blinkt in der Bildschirmmitte die Meldung, daß die Übersetzung erfolgreich verlaufen ist. Mit einer beliebigen Taste bestätigen Sie die Meldung und kehren zurück in das Editfenster.

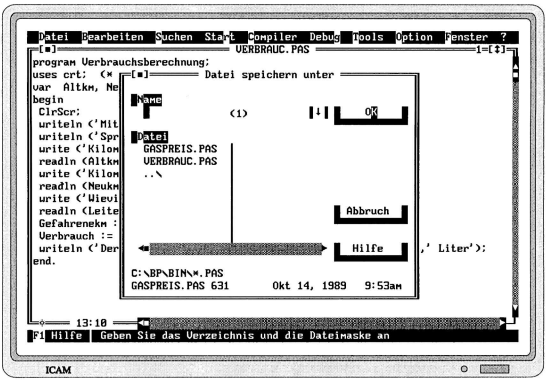
Weist das Programm aber noch syntaktische Fehler auf, schaltet das System in das Editfenster. Am oberen Rand wird eine Fehlermeldung angezeigt, die den Fehler näher beschreibt. Der Cursor zeigt im Quellprogramm die Stelle an, an der der Compiler den Fehler festgestellt hat.



Mittels Esc bestätigen Sie die Fehlermeldung. Wie Syntaxfehler zu behandeln sind, wird im Kapitel 3.6 dargestellt.

Programm speichern

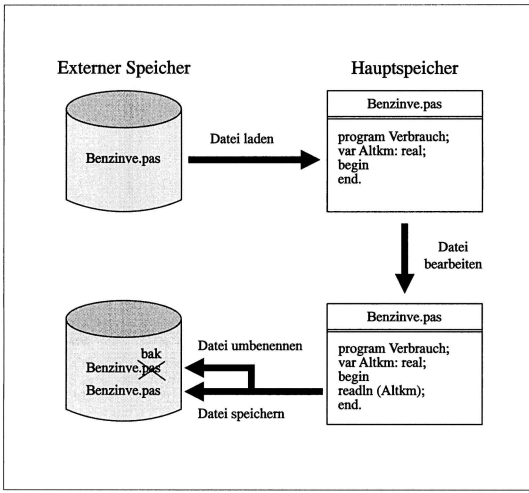
Damit das eingegebene Programm nicht infolge fehlerhafter Rechnerbedienung verlorengeht, sollten Sie es zwischenzeitlich auf der Diskette oder Festplatte speichern. Die Speicherungsfunktion rufen Sie mit F2 auf.



In dem eingeblendeten Fenster geben Sie in der Eingabezeile (1) den Namen der Datei an, in der Ihr Programm gespeichert werden soll. Der Dateiname wird entsprechend den Regeln des DOS-Betriebssystems aus bis zu 8 Zeichen gebildet; er darf aus

- den Buchstaben A–Z,
- den Ziffern 0–9 oder
- den Sonderzeichen ! ( ) @ ^ # - _ ´ ` $ % & ~ { }

bestehen. Die Eingabe des Dateinamens ist mit ↵ oder Alt K abzuschließen. Als Erweiterung hängt das Pascal-System automatisch .PAS an den Dateinamen an.
Speichern Sie Ihr Programm zum wiederholten Mal, so legt das System automatisch eine .BAK-Datei an. Während die .PAS-Datei jeweils den aktuellen Bearbeitungsstand enthält, beinhaltet die .BAK-Datei den jeweils vorletzten Stand der Bearbeitung.



Ausführen des Programms

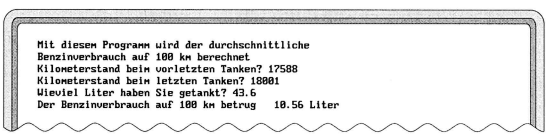
Ein syntaktisch fehlerfreies Programm wird aus dem Menüpunkt *Start* bzw. *Run* gestartet. Dieselbe Wirkung hat die Tastenkombination Strg F9. Bei der Programmausführung schaltet das System auf den Ausgabebildschirm um, um dort Eingaben entgegenzunehmen oder Ergebnisse anzuzeigen. Nach dem Start des Programms Verbrauchsberechnung erscheint auf dem Bildschirm:

```
Mit diesem Programm wird der durchschnittliche
Benzinverbrauch auf 100 km berechnet
Kilometerstand beim vorletzten Tanken?
```

Geben Sie nun die Werte ein. Jede Eingabe ist mit ↵ abzuschließen.

Beispiel:
Kilometerstand beim vorletzten Tanken? 17588
Kilometerstand beim letzten Tanken? 18001
Wieviel Liter haben Sie getankt? 43.6

Hinweis: Bei der Eingabe von Dezimalzahlen wird der **Dezimalpunkt**, nicht aber das Dezimalkomma verwendet. Auch werden nur die Zahlenwerte, nicht aber die Bezeichnungen wie l für Liter oder km für Kilometer eingegeben. Ergebnis der Berechnung:



Das Bild ist jedoch kaum zu sehen, da das System unmittelbar nach Beendigung des Programmlaufs wieder in das Editfenster zurückschaltet. Um die Datenausgabe betrachten zu können, stellen Sie mittels Alt F5 auf den Ausgabebildschirm um. Das Anschlagen einer beliebigen Taste bringt Sie zurück in den Editbildschirm.
Wenn Ihnen das Umschalten mittels Alt F5 lästig ist, so sollten Sie das Ausgabefenster einrichten, in dem Sie die Datenausgabe betrachten können.

Drucken eines Programms
Um das Programm, das Sie gerade bearbeiten, ausdrucken zu lassen, rufen Sie den Auswahlpunkt *Drucken* im Menü *Datei* auf.

1.1.5 Zusammenfassung

- Das Bediensystem wird über die Menüleiste und die Funktionstastenleiste gesteuert.
- Ein Auswahlfeld in der Menüleiste wird entweder durch Anklicken mit der Maus, durch die Tastenkombination Alt und Buchstabe oder durch die Tastenfolge F10, Buchstabe angewählt. Der anzuschlagende Buchstabe ist in dem Menüpunkt hervorgehoben dargestellt.

- Die Funktionen des Bediensystems werden über Auswahlpunkte in Pull-Down-Menüs oder über Funktionstasten aufgerufen.
- Der Nutzer kann zu dem Editfenster das Überwachungs- und das Ausgabefenster einrichten.
- Die Fenster können vergrößert, verkleinert und hinsichtlich ihrer Lage auf dem Bildschirm verändert werden.
- Es können mehrere Editfenster aufgebaut werden; diese werden jeweils numeriert.
- Ein Fenster kann durch [Alt] und Fensternummer oder durch Anklicken seines Rahmens oder durch [F6] aktiviert werden.
- Ein aktives Fenster wird durch [Alt] [F3] aufgehoben.
- Die Festlegung des aktuellen Verzeichnisses und des Programmnamens erfolgt in Dialogfenstern.
- In einem Dialogfenster werden Eingabezeilen und Aktionsschalter mit dem Mauszeiger oder der [↹]-Taste oder durch die Tastenkombination [Alt] und hervorgehobener Buchstabe aktiviert.
- Ein Verzeichnisname oder Programmname darf aus bis zu acht Zeichen, Ziffern oder Sonderzeichen bestehen.
- Beim Speichern eines Programms hängt das Turbo-Pascal-System automatisch die Kennung .PAS an den Dateinamen an. Eine bereits bestehende namensgleiche .PAS-Datei erhält die Kennung .BAK.
- Der Compiler übersetzt das Quellprogramm in das ausführbare Maschinenprogramm.
- Nur ein syntaktisch korrektes Programm kann ausgeführt werden.

1.1.6 Übungen

1. Aufgabe
Wie kann der Menüpunkt *Compiler* aufgerufen werden?

2. Aufgabe
Mit welcher Taste wird ein Pull-Down-Menü oder ein Dialogfenster aufgehoben?

3. Aufgabe
Woran erkennt man, welches Fenster aktiv ist?

4. Aufgabe
Wie kann ein Fenster aktiviert werden?

5. Aufgabe
Sie wollen ein Fenster verkleinern und in die rechte obere Bildschirmecke verschieben. Wie gehen Sie vor?

6. Aufgabe
Warum empfiehlt es sich, zu Beginn einer Arbeitssitzung das aktuelle Verzeichnis festzulegen?

7. Aufgabe
Auf welchem Weg kann man das Pascal-System kurzzeitig verlassen, um auf der Betriebssystemebene zu arbeiten? Über welches Kommando kehrt man in das Pascal-System zurück?

8. Aufgabe
Warum ist der Dateiname VERSU 1 falsch?

9. Aufgabe
Welche Aufgabe hat ein Compiler?

10. Aufgabe
Normalerweise werden im Dialogfenster *Datei öffnen* nur die .PAS-Dateien angezeigt. Sie möchten auf eine .BAK-Datei zurückgreifen. Wie führen Sie den Ladevorgang durch?

1.1.7 Lösungen

1. Aufgabe
Der Menüpunkt *Compiler* kann wie folgt aufgerufen werden:
(a) Anklicken des Begriffs *Compiler* in der Menüleiste.
(b) Aktivieren der Menüleiste durch F10, anschließend C eingeben.
(c) Anwahl durch die Tastenkombination Alt C.
(d) Aktivieren der Menüleiste, den Cursorbalken mittels ← oder → auf *Compiler* stellen. Wahl mit ↵ bestätigen.

2. Aufgabe
Die Taste [Esc] hebt eingeblendete Pull-Down-Menüs und Dialogfenster auf.

3. Aufgabe
Ein aktives Fenster ist durch einen Doppelrahmen gekennzeichnet.

4. Aufgabe
Zum Aktivieren kann man mehrere Wege einschlagen:
(a) Anklicken des Rahmens mittels der Maus
(b) Eingabe der Tastenkombination [Alt] und Fensternummer.
(c) Betätigen der Taste [F6] oder [⇧] [F6].

5. Aufgabe
Zunächst das Fenster aktivieren, die *Fenster/Größe/Position*-Funktion durch [Strg] [F5] anschalten und durch [⇧] und Pfeiltasten das Fenster an die gewünschte Position verschieben. Bei Einsatz einer Maus wird die rechte untere Ecke angeklickt und das Fenster entsprechend der Mausbewegung verkleinert und verschoben.

6. Aufgabe
Unterläßt man die Einstellung eines eigenen Unterverzeichnisses, so werden die Programme in dem Verzeichnis gespeichert, aus dem das Turbo-Pascal-System geladen wurde. Das Verzeichnis würde dann sehr bald unübersichtlich, was die Arbeit erschwert.

7. Aufgabe
Man ruft im *Datei*-Menü die Funktion *DOS aufrufen* auf. Mittels **exit** [↵] kehrt man in das Pascal-System zurück.

8. Aufgabe
Ein Dateiname darf nur aus Ziffern, Buchstaben und den Sonderzeichen ! ( ) @ ^ # - _ ´ ` $ % & ~ { } bestehen. Er darf kein Leerzeichen enthalten.

9. Aufgabe
Der Compiler übersetzt ein vom Programmierer eingegebenes Quellprogramm in das vom Computer ausführbare Maschinenprogramm.

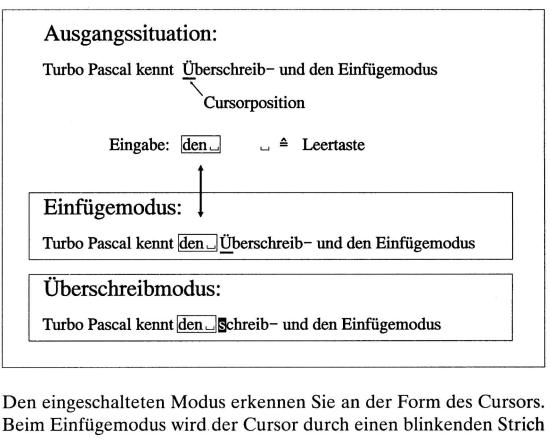
10. Aufgabe
Man gibt im Namensfeld des Dialogfensters *.BAK ↵ ein. Unter *Datei* werden nun die Namen der .BAK-Dateien angezeigt.

1.2 Editorsystem

Bei der Eingabe und der Bearbeitung eines Programmtextes unterstützt das Editorsystem den Programmierer durch eine Vielzahl von Editierkommandos. Die wichtigsten werden in den folgenden Abschnitten dargestellt.

1.2.1 Überschreib- und Einfügemodus

Die Eingabe bzw. Bearbeitung eines Programmtextes kann im Einfüge- oder im Überschreibmodus erfolgen. Steht der Cursor innerhalb einer Textzeile, so wird beim Einfügemodus der rechts vom Cursor befindliche Text bei einer Texteingabe nach rechts geschoben. Beim Überschreibmodus dagegen wird der Text überschrieben.



Den eingeschalteten Modus erkennen Sie an der Form des Cursors. Beim Einfügemodus wird der Cursor durch einen blinkenden Strich

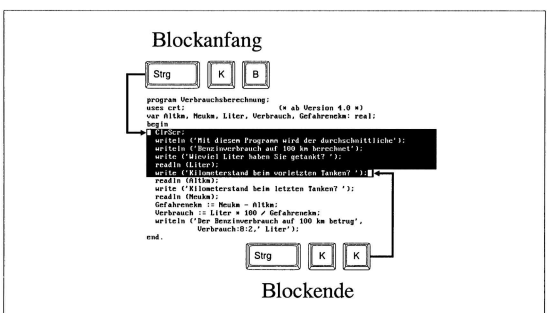
dargestellt. Im Überschreibmodus nimmt der Cursor die Form eines blinkenden Blocks an. Durch die Taste [Einfg] schalten Sie von einem Modus in den anderen um.

1.2.2 Blockoperationen

Möchten Sie eine Folge von Zeichen oder Zeilen kopieren oder löschen, so sind diese zunächst als Block zu markieren. Zur Markierung eines Blocks stehen Ihnen verschiedene Möglichkeiten offen.

Blöcke markieren

Die in früheren Versionen von Turbo-Pascal anzuwendenden Kommandos sind auch in der Version 7 verfügbar. Sie können also den Text mit [Strg] [K] [B] und [Strg] [K] [K] markieren. Dazu stellen Sie den Cursor auf den Anfang des zu markierenden Textes und betätigen [Strg] [K] [B]. Anschließend setzen Sie den Cursor unmittelbar hinter den zu markierenden Text und geben die Tastenkombination [Strg] [K] [K] ein.



Der markierte Bereich wird farblich hervorgehoben. Zur Erinnerung: Die Tastenkombination [Strg] [K] [B] wird in der Weise eingegeben, daß Sie die Tasten [K] und [B] anschlagen, während Sie die Taste [Strg] niederhalten. Anschließend lassen Sie die [Strg]-Taste los. Die Tastenkombination [Strg] [K] [H] hebt die Markierung wieder auf.

Auch setzt die Markierung eines anderen Bereichs die bisherige Markierung außer Kraft. Es kann also stets nur ein Bereich markiert sein.

Ab der Version 6 stehen noch weitere Markierungsfunktionen zur Verfügung. Wollen Sie eine Zeile markieren, so gehen Sie wie folgt vor:

(a) Mit Maus: Stellen Sie den Mauszeiger in die betreffende Zeile, und betätigen Sie die linke Maustaste zweimal.

(b) Ohne Maus: Stellen Sie den Cursor in die betreffende Zeile, und betätigen Sie die Tastenkombination [Strg] [K] [L].

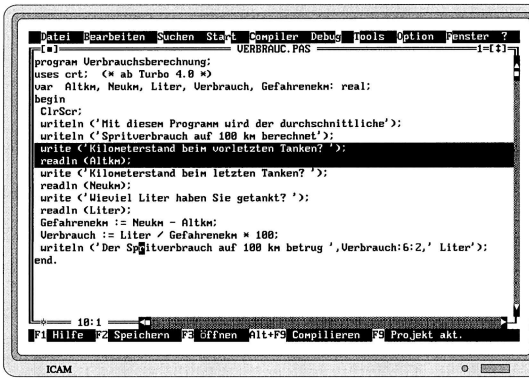
Mehrere aufeinander folgende Zeilen können Sie wie folgt als Block markieren:

(a) Mit Maus: Stellen Sie den Mauszeiger auf den Beginn des Textes, und ziehen Sie ihn mit niedergehaltener Maustaste über den Text. Der markierte Text wird farblich hervorgehoben.

(b) Ohne Maus: Stellen Sie den Cursor auf den Beginn des Textes. Halten Sie die [⇧]-Taste nieder, und fahren Sie den Cursor mit Hilfe der Cursortasten über den zu markierenden Textbereich.

Beispiel:

Setzen Sie den Cursor an den Anfang der Zeile 8, und betätigen Sie die Tastenkombination [⇧] [↓]. Der markierte Bereich wird mit einem veränderten Hintergrund dargestellt.



Zeilen oder Blöcke löschen

Beim Löschen von Programmteilen sind zwei Fälle zu unterscheiden. Im ersten Fall wird der betreffende Textteil nicht mehr benötigt. Im zweiten Fall muß er von der jetzigen Stelle weggenommen, später aber an einer anderen Stelle wieder eingefügt werden.

Unwiederbringliches Löschen: Wollen Sie nur eine Zeile löschen, so stellen Sie den Cursor in die Zeile und betätigen die Tastenkombination Strg Y. Das Kommando zum Löschen eines markierten Bereichs (Block) lautet Strg K Y. Dieselbe Wirkung hat Strg Entf.

Löschen mit Kopie: Möchten Sie eine oder mehrere Zeilen löschen, die Sie an einer anderen Stelle weiterverwenden wollen, so markieren Sie zunächst den Textbereich und kopieren ihn mit ⇧ Entf in die **Zwischenablage** bzw. in das **Clipboard**. Die Zwischenablage ist ein weiteres Fenster, das vom System automatisch geführt wird. Es dient als Zwischenspeicher, der auch dann erhalten bleibt, wenn Sie das Programm wechseln.

Beispiel:

In dem obigen Programm soll die Reihenfolge der Eingaben geändert werden. Erst sollen die Kilometerangaben, dann die Literanga-

be gemacht werden. Wenn Sie in dem Programmtext die beiden Zeilen – wie oben abgebildet – als Block markiert haben, so löschen Sie mit der Tastenkombination ⇧ Entf den Block in dem Programm und kopieren ihn automatisch in die Zwischenablage. Der zuletzt in das Ablagefenster kopierte Text ist durch eine Blockmarkierung hervorgehoben.

Über das Hauptmenü können Sie wie folgt auf das Ablagefenster zugreifen:

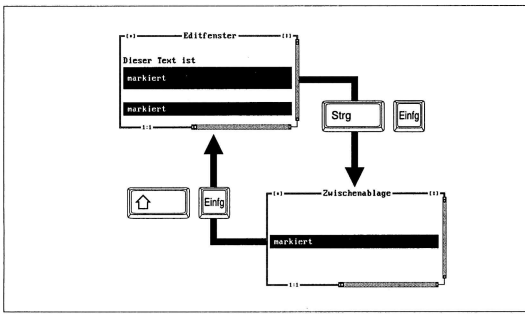
Version 7	Version 6
Bearbeiten/Zwischenablage anzeigen	Edit/Show Clipboard

F6 schaltet ebenfalls ins Ablagefenster um. Mit ⇧ F6 gelangen Sie wieder zurück in das Editfenster.
Um Text aus der Zwischenablage in das aktive Fenster zu kopieren, betätigen Sie die Tastenkombination ⇧ Einfg.

Beispiel:
Sie wollen in dem obigen Beispiel den gelöschten Text der ehemaligen Zeilen 8 und 9 aus der Zwischenablage hinter die Zeile 13 kopieren. Stellen Sie den Cursor auf den Beginn der Zeile 14, und geben Sie die Tastenkombination Strg N ein. Damit schaffen Sie sich eine Leerzeile. Mittels ⇧ Einfg wird der Textblock aus der Zwischenablage an die neue Stelle geschrieben.

Blöcke kopieren
Möchten Sie eine oder mehrere zusammenhängende Zeilen an andere Stellen kopieren, so markieren Sie zunächst die Zeile oder den Block. Strg Einfg bildet den markierten Bereich im Ablagefenster ab. Anschließend fahren Sie den Cursor an die Stelle im Programm,

auf die die Kopie erfolgen soll. ⇧ Einfg überträgt die Kopie aus der Zwischenablage in das Programm. Durch das Kopieren wird der Eintrag im Ablagefenster nicht verändert.



Während einer Arbeitssitzung sammelt sich eine Reihe von Kopien im Ablagefenster an. Da stets der letzte Eintrag markiert ist, sind die übrigen Einträge nicht mehr mit einer Markierung versehen. Um einen früheren Eintrag in das Programm im Editfenster zu kopieren, gehen Sie wie folgt vor:

1. Schalten Sie um in das Ablagefenster, und markieren Sie dort den betreffenden Text als Block.
2. Aktivieren Sie das Editfenster, und stellen Sie den Cursor an die aufnehmende Stelle. Schaffen Sie sich bei Bedarf durch Strg N eine Leerzeile.
3. Kopieren Sie mittels ⇧ Einfg den markierten Text an die aktuelle Cursorposition.

Programmteile kopieren

In Verbindung mit dem Ablagefenster wird auch das Kopieren eines Blocks aus einem Programm in ein anderes stark vereinfacht.

Beispiel:

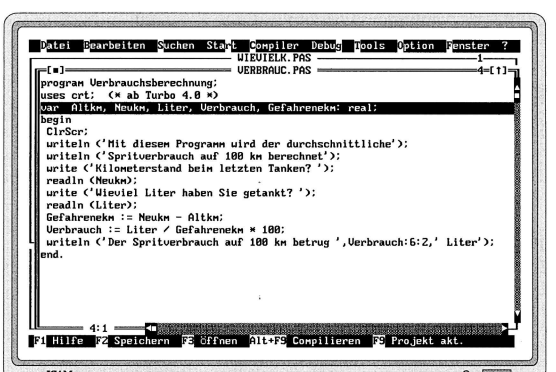
Sie wollen ein Programm schreiben, mit dem berechnet wird, bei wel-

chem Kilometerstand wieder getankt werden muß. Bei dieser Aufgabenstellung können Sie einige Teile aus dem Programm Verbrauchsberechnung, das unter VERBRAUC.PAS gespeichert ist, übernehmen. Sie haben bereits mit dem Programm WIEVIELK.PAS angefangen und wollen nun die Variablenvereinbarung aus dem Programm VERBRAUC.PAS herüberkopieren.

Vorgehensweise:

1. Laden Sie das Programm VERBRAUC.PAS mittels F3. Das Programm wird in einem weiteren Fenster am Bildschirm angezeigt.
2. Markieren Sie die Variablenvereinbarungszeile, und kopieren Sie sie mittels Strg Einfg in das Ablagefenster.
3. Aktivieren Sie das aufnehmende Editfenster, beispielsweise durch Alt 1, und stellen Sie den Cursor an die gewünschte Position.
4. Übertragen Sie den Textblock aus dem Ablagefenster mit ⇧ Einfg.

Zu Schritt 1 und 2:



Zu Schritt 3 und 4:

Strg Einfg kopiert aus einem Editfenster in das Ablagefenster.
⇧ Einfg kopiert aus dem Ablagefenster in das aktive Editfenster.

Block in Datei schreiben

Wird eine Anweisungsfolge in mehreren Programmen benötigt, so sollte man diese in einer eigenen Datei auf der Platte oder Diskette speichern. Von dort können Sie den Anweisungsblock bei Bedarf in andere Programme hineinkopieren.

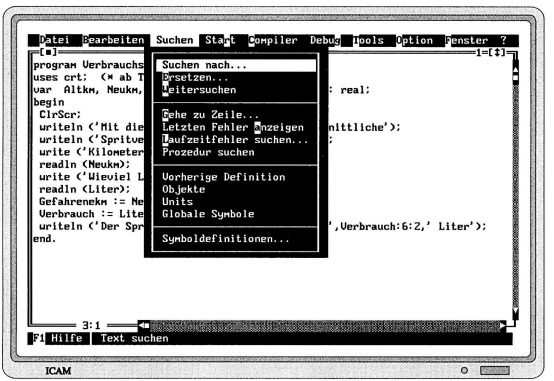
Zunächst fassen Sie die Anweisungsfolge zu einem Block zusammen. Mit dem Kommando Strg K W (für Write) rufen Sie das Fenster Schreiben Block in Datei auf, in das Sie den Dateinamen eintragen. Die Bedienweise dieses Dialogfensters entspricht der des Speichern-Fensters, das oben bereits behandelt wurde.

Block aus Datei lesen

Mittels Strg K R (für Read) wird der Inhalt einer Datei ab der Cursorstelle in das Programm kopiert, das Sie im aktiven Fenster bearbeiten.

1.2.3 Suchen und Ersetzen

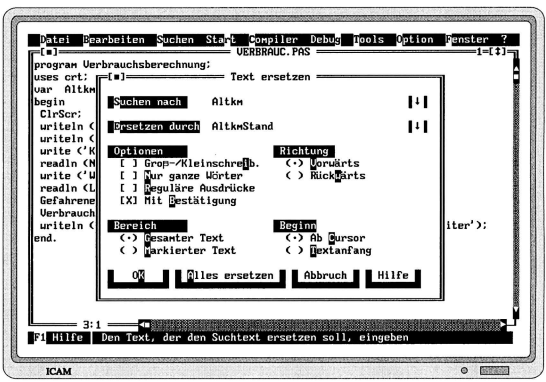
Die Suchen-und-Ersetzen-Funktion ist so wichtig, daß ihr ab der Version 6 ein eigener Menüpunkt gewidmet wird. Arbeiten Sie mit einer früheren Version, müssen Sie die Kommandos von Turbo-Pascal 3 anwenden.



Eingabe der Begriffe

Verdeutlichen wir uns die Ersetzungsfunktion an Hand eines Beispiels. In dem Programm Verbrauchsberechnung soll der Name Altkm durch AltkmStand ersetzt werden. Positionieren Sie den Cursor auf Altkm, und rufen Sie im Menü *Suchen* den Unterpunkt *Ersetzen* auf. Das System öffnet ein Dialogfenster, in dem Sie in der Eingabezeile *Suchen nach* bereits «Altkm» als Suchbegriff vorfinden. Falls Sie den Suchbegriff nicht aus dem Editfenster übernehmen, geben Sie ihn in dieser Zeile ein. Sie können ihn aber auch gegebenenfalls aus der Eingabeaufzeichnungsliste entnehmen.
Wenn Sie keine Maus verwenden, verlassen Sie die Eingabezeile mittels ⇆, nicht durch ↵. Bei Mausverwendung klicken Sie *Ersetzen durch* an. Damit öffnen Sie die Eingabezeile, in der Sie den Begriff eingeben, der das Suchwort ersetzen soll. Auch hier können Sie den

Ersatzbegriff aus der Eingabeaufzeichnungsliste abrufen. Wenn Sie die Suchbedingungen verändern wollen, schließen Sie die Eingabezeile mittels ⇆ ab. Bei Mauseinsatz klicken Sie die zu verändernde Suchbedingung an. Ansonsten starten Sie den Such- und Ersetzungsvorgang mittels ↵ oder durch Betätigen der Aktionsschalter *OK* oder *Alles ersetzen*.



In dem Dialogfenster geben Sie ferner die Bedingungen an, unter denen der Such- und Austauschvorgang erfolgen soll. Dazu stehen Ihnen das Markierungsfeld *Optionen* und die Schaltfelder *Bereich*, *Richtung* und *Beginn* zur Verfügung.

Markierungsfeld *Optionen*

In dem Markierungsfeld *Optionen* können Sie mehrere Bedingungen setzen. Mittels der Pfeiltasten steuern Sie die einzelnen Bedingungszeilen an. Eine Bedingung wird gesetzt bzw. zurückgenommen durch Betätigen der Leertaste. Eine vereinbarte Bedingung ist an [X] zu erkennen.

Im Markierungsfeld *Optionen* bedeuten:

Option	Version 7	Version 6	Bedeutung
[X]	Groß-/Kleinschreib.	Case sensitive	Es wird zwischen Groß- und Kleinschreibung unterschieden
[X]	Nur ganze Wörter	Whole words only	Nur ganze Wörter werden beim Ersetzen berücksichtigt

Beispiel:
Es soll das Wort renr (Abkürzung von Rechnungsnummer) durch RechNr ersetzt werden. Ohne den Schalter *Nur ganze Wörter* würde auch das Wort Tourenr geändert in TouRechNr.

Option	Version 7	Version 6	Bedeutung
[X]	Reguläre Ausdrücke	Regular expression	Suche gemäß Textmuster

Ist die Bedingung *Reguläre Ausdrücke* gesetzt, erfolgt die Suche auf der Grundlage eines Textmusters. Für die Beschreibung der Textmuster werden sogenannte Jokerzeichen eingesetzt, von denen die wichtigsten hier im Zusammenhang mit dem Programm Verbrauchsberechnung vorgestellt werden.

.Km — findet alle Begriffe, in denen die Buchstabenfolge **Km** vorkommt. Der Punkt steht für ein beliebiges Zeichen.

Verb* — findet alle Begriffe, in denen die Zeichenfolge **Ver** auftritt. Mit b* wird festgelegt, daß ab dieser Stelle beliebige Zeichen zugelassen sind. * steht nach einem Zeichen und ersetzt beliebig viele. Bei dieser Suchbedingung wird auch der Begriff Version gefunden.

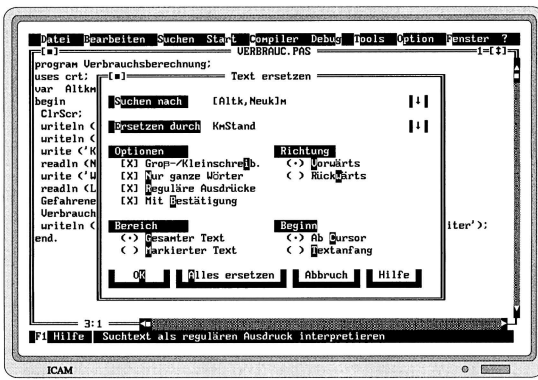
Verb+ — findet alle Begriffe, in denen die Zeichenfolge **Verb** vorkommt. Das + besagt, daß die vorstehende Zeichenfolge unbedingt im Suchbegriff enthalten sein muß. Der Begriff Version wird bei dieser Bedingung nicht gefunden.

[t,u]Km — Es werden alle Begriffe gefunden, in denen **tKm** oder **uKm** vorkommen. Der Begriff GefahreneKm wird nicht gefunden, weil der Zeichenfolge Km ein e vorausgeht.

[e-t]Km findet alle Begriffe, in denen die Muster **eKm**, **fKm** bis **tKm** vorkommen. NeuKm wird nicht gefunden, weil u nicht zu der Buchstabenmenge e – t gehört.

Beispiel:

Es soll Altkm bzw. Neukm durch AltKmStand bzw. NeuKmStand ausgetauscht werden. Um beide Suchbegriffe in einem Ersetzungslauf zu bearbeiten, wird auf die Möglichkeit von *Reguläre Ausdrücke* zurückgegriffen. Die einzugebenden Begriffe lauten dann:



Option	Version 7	Version 6	Bedeutung
[X]	Mit Bestätigung	Prompt on replace	Das Ersetzen erfolgt erst auf Rückfrage

Sie verlassen das Optionenfeld durch eine entsprechende Mausbewegung oder durch ↹.

Für das obige Beispiel, in dem Altkm durch AltKmStand ersetzt werden soll, könnte man die Bedingungen wie folgt setzen:

Option	Version 6	Version 7
[X]	Case sensitive	Groß-/Kleinschreibung
[X]	Whole words only	Nur ganze Wörter
[ ]	Regular expression	Reguläre Ausdrücke
[ ]	Prompt on replace	Mit Bestätigung

In der Version 5.x ist die Such- und Ersetzfunktion wie in der Version 3 geregelt (s. Kap. 8.3).

Schaltfelder
Schaltfelder sind durch runde Klammern gekennzeichnet. Sie können dort eine von mehreren Bedingungen auswählen. Mittels einer Pfeiltaste oder durch Anklicken schalten Sie von einer Möglichkeit zur anderen. Sie verlassen ein Schaltfeld durch ↹ oder durch eine entsprechende Mausbewegung.
Im Schaltfeld *Richtung* ist anzugeben, ob der Text in Richtung Dateiende (*Vorwärts*) oder Dateianfang (*Rückwärts*) durchsucht werden soll. In *Bereich* wird der Gültigkeitsbereich festgelegt. *Gesamter Text* bedeutet, daß sich die Operation auf die gesamte Datei beziehen soll. Anderenfalls ist die Gültigkeit auf den markierten Bereich beschränkt. Im Schaltfeld *Beginn* wird der Startpunkt festgelegt. Ist der Schalter *Ab Cursor* gesetzt, so beginnt die Such- und Ersetzfunktion an der Cursorstelle. Ansonsten wird der gesamte Gültigkeitsbereich angesprochen.

Aktionsschalter
Der Such- und Ersetzungsvorgang wird durch die Aktionsschalter *OK* oder *Alles ersetzen* ausgelöst. Der Unterschied zwischen diesen Aktionsschaltern liegt darin, daß bei *OK* zunächst nur eine Ersetzung vorgenommen wird. Durch *Alles ersetzen* wird dagegen ausgedrückt, daß sich der Ersetzungsvorgang auf den ganzen Gültigkeitsbereich beziehen soll. Mit der Tastenkombination Strg L können Sie die Such- bzw. Ersetzungsoperation fortsetzen, wenn der Aktionsschalter *OK* betätigt wurde.

Suchen
Über die Funktion *Suchen* kann nach einem bestimmten Begriff oder nach bestimmten Zeichenmustern gesucht werden. Das Dialog-

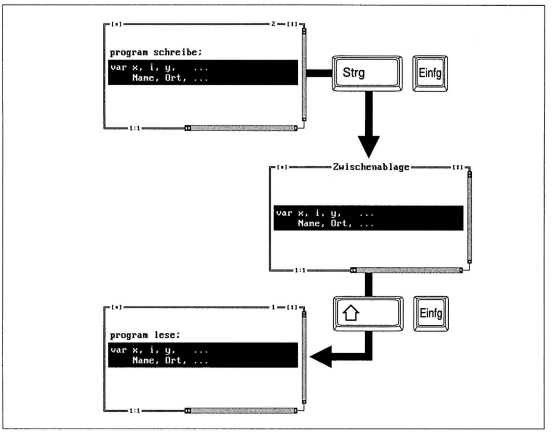
fenster zu dieser Funktion entspricht in seinem Aufbau dem Suchen-und-Ersetzen-Fenster.

1.2.4 Zusammenfassung

- [Einfg] schaltet vom Einfügemodus in den Überschreibmodus und umgekehrt.
- Ein Block besteht aus einer Folge von Zeichen, die markiert worden sind.
- Blockmarkierungen können durch
  a) [Strg] [K] [B] und [Strg] [K] [K]
  b) [⇧] und Pfeiltasten
  c) Mausbewegungen
  vorgenommen werden.
- Löschkommandos:
  Es kann zeichen-, wort-, zeilen- und blockweise gelöscht werden.

Funktion	Taste
Zeichen links vom Cursor löschen	[←]
Zeichen auf Cursorposition löschen	[Entf]
Wort rechts vom Cursor löschen	[Strg] [W]
Zeile, in der der Cursor steht, löschen	[Strg] [Y]
Rest einer Zeile, der rechts vom Cursor steht, löschen	[Strg] [Q] [Y]
Block ohne Zwischenspeicherung löschen	[Strg] [Entf] oder [Strg] [K] [Y]
Block mit Zwischenspeicherung in der Zwischenablage löschen	[⇧] [Entf]

- Die Zwischenablage ist ein Fenster, das als Zwischenspeicher dient.
- Mit den Tastenkombinationen [Strg] [Einfg] und [⇧] [Einfg] wird auf das Ablagefenster zugegriffen.



- Mittels Strg K W wird ein Block in eine Datei geschrieben.
- Strg K R liest eine Datei in das aktive Editfenster.
- Strg N schafft eine Leerzeile.
- In Dialogfenstern werden die Eingaben, die der Nutzer gemacht hat, in einer Aufzeichnungsliste vermerkt. Sie können aus der Liste wieder aufgerufen werden.

1.2.5 Übungen

1. Aufgabe
Woran erkennt man, ob sich der Editor im Einfüge- oder im Überschreibmodus befindet?

2. Aufgabe
Wie bildet man aus mehreren aufeinanderfolgenden Zeilen mit Hilfe der Pfeiltasten einen Block?

3. Aufgabe
Wie wird eine Markierung aufgehoben?

4. Aufgabe
Welcher Unterschied besteht zwischen den beiden Löschkommandos [Strg] [Entf] und [⇧] [Entf]?

5. Aufgabe
Welche Wirkung hat [Strg] [Einfg], welche hat [⇧] [Einfg]?

6. Aufgabe
Eine Anweisungsfolge, die eine Eingabemaske auf dem Bildschirm aufbaut, wird in mehreren Programmen benötigt. Wie verfahren Sie, um diese Anweisungsfolge nicht jeweils neu schreiben zu müssen?

7. Aufgabe
Sie möchten in einem Programmtext den Variablennamen Kante ersetzen durch Seite. In dem Programm werden auch noch die Namen AufKante und AbKante verwendet. Wie müssen Sie die Suchbedingungen formulieren?

1.2.6 Lösungen

1. Aufgabe
Im Überschreibmodus erscheint der Cursor als blinkender Block.

2. Aufgabe
Man stellt den Cursor auf den Anfang der ersten Zeile. Durch Anschlagen und Festhalten der Tastenkombination [⇧] [↓] wird die Markierung vorgenommen.

3. Aufgabe
Eine Markierung wird durch [Strg] [K] [H] oder durch Markierung eines anderen Bereichs aufgehoben.

4. Aufgabe
[Strg] [Entf] löscht wie [Strg] [K] [Y] einen Block endgültig. Beim Kommando [⇧] [Entf] wird der Block aus dem Editfenster entfernt und im Ablagefenster (Clipboard) zwischengespeichert.

5. Aufgabe
[Strg] [Einfg] kopiert einen Block aus dem aktiven Editfenster in das

Ablagefenster. ⇧ Einfg kopiert einen Block aus dem Ablagefenster an die Cursorposition im aktiven Editfenster.

6. Aufgabe

Die Anweisungsfolge wird als Block markiert und mittels Strg K W in eine Datei geschrieben, aus der sie bei Bedarf mittels Strg K R an die Cursorposition im aktiven Editfenster geladen werden kann.

7. Aufgabe

Die Bedingung
[X] *Nur ganze Wörter*
muß gesetzt sein.

2 AUFBAU EINES PASCAL-PROGRAMMS

Jedes Pascal-Programm muß nach bestimmten Regeln (= Syntax) aufgebaut werden. Werden diese Regeln nicht eingehalten, ist das Programm fehlerhaft, und der Computer kann es nicht verarbeiten. An Hand des Programms zur Verbrauchsberechnung werden die Regeln zum Programmaufbau dargelegt.
Laden Sie dazu das Programm VERBRAUC.PAS, und lassen Sie es sich auf dem Bildschirm anzeigen.

Bedienhinweise:

Arbeitsablauf	**Kommando**		
	Version 3	**Versionen 4, 5, 6**	**Version 7**
Laufwerk und/oder Verzeichnis einstellen	L A	F10 F C	F10 D V
Aufruf des Ladekommandos	W	F3	
Eingabe des Dateinamens	Verbrauc.pas ↵	Auswahl aus Verzeichnis	
Aufruf Editor	E		

```
0  program Verbrauchsberechnung;
   uses crt;                                    (* ab Version 4 *)
1  var  Altkm, Neukm, Liter, Verbrauch, Gefahrenekm: real;
2  begin
3    ClrScr;
4    writeln ('Mit diesem Programm wird der durchschnittliche');
5    writeln ('Benzinverbrauch auf 100 km berechnet');
6    write ('Kilometerstand beim vorletzten Tanken? ');
7    readln (Altkm);
8    write ('Kilometerstand beim letzten Tanken? ');
9    readln (Neukm);
10   write ('Wieviel Liter haben Sie getankt? ');
11   readln (Liter);
12   Gefahrenekm := Neukm - Altkm;
13   Verbrauch := Liter / Gefahrenekm * 100;
14   writeln ('Der Benzinverbrauch auf 100 km betrug ',
               Verbrauch:8:2,' Liter');
15 end.
```

In dem obigen Programmtext sind den Zeilen Nummern vorangesetzt. Sie dienen lediglich der besseren Orientierung bei den nachstehenden Erklärungen.

2.1 Programmkopf

Jedes Pascal-Programm beginnt mit einem Programmkopf. Eingeleitet wird es mit dem Schlüsselwort «program». Schlüsselwörter sind feststehende Begriffe, die in der Programmiersprache festgelegt worden sind. Hinter einem Schlüsselwort folgt in der Regel eine Leerstelle. Dahinter wird der Name des Programms als Überschrift angegeben. Der Name kann beliebig gewählt werden. Dabei gelten folgende Regeln:

(1) Ein Name besteht aus einer Folge von
    Buchstaben
    Ziffern
    Sonderzeichen «_» (Unterstrich)
(2) Das erste Zeichen darf keine Ziffer sein.
(3) Ein Name darf aus höchstens 127 Zeichen bestehen.

Beispiele für Namen:

MWST15	richtig
MWST 15	falsch, da ein Leerzeichen vorkommt
MWST_15	richtig

Der Programmkopf wird abgeschlossen durch ein Semikolon. Allgemein lautet der Programmkopf in Turbo-Pascal

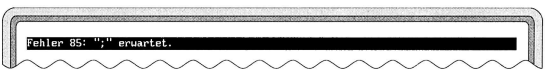
program Name;

Das Semikolon hat in Pascal eine besondere Bedeutung. Es trennt eine Anweisung von der nächsten.

Beispiel:
Fahren Sie den Cursor auf das Semikolon im Programmkopf (Zeile 0), und radieren Sie es mittels [Entf] aus! Starten Sie den Compiler:

Version 3	ab Version 4
[Strg] [K] [D]	[F10] [C]
[C]	[C]

Das System meldet den Fehler.



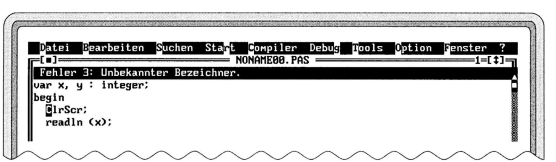
Drücken Sie die Taste [Esc]. Das System schaltet wieder auf den Editor um und zeigt das Programm am Bildschirm an. Der Cursor steht am Beginn der Zeile 1. Dort hat er festgestellt, daß die vorherige Anweisung nicht richtig abgeschlossen worden ist. Fahren Sie deshalb den Cursor wieder an das Ende der Zeile 0, und ergänzen Sie das Semikolon.

Das Semikolon trennt eine Anweisung von der nächsten.

Im Gegensatz zu anderen Pascal-Dialekten brauchen in Turbo-Pascal die im Programm benötigten Dateien nicht angegeben zu werden. Wenn Sie aber Programme entwickeln wollen, die auch unter anderen Pascal-Dialekten laufen sollen, müssen Sie den Programmkopf wie folgt formulieren:

program Name **(input,output);**

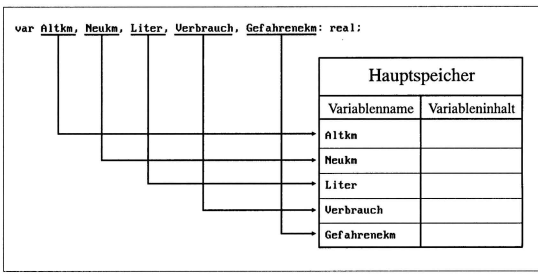
Input bezeichnet die Standard-Eingabedatei (Tastatur). Output ist der Name der Standard-Ausgabedatei (Bildschirm).
Die Anweisung **uses crt;** ist eine Besonderheit des Turbo-Pascal-Systems. Beim Aufbau des Maschinenprogramms muß das System eine Vielzahl von Prozeduren und Funktionen zusammenstellen. Diese Routinen sind ab der Version 3 nicht zusammenhängend in einer Datei, sondern in sachgebietsbezogenen **Units** (engl. Einheiten) zusammengefaßt. Beispielsweise sind die Maschinenbefehle für die Anweisung ClrScr (**ClearScr**een = Bildschirm löschen) in der Unit **CRT** enthalten. CRT ist die Abkürzung für cathod-ray-tube (= Kathodenstrahlröhre) und steht im englischen Sprachgebrauch für den Begriff Bildschirm.
Damit die für die ClrScr-Anweisung erforderlichen Maschinenbefehle in das Programm eingebunden werden, muß man dem Pascal-System durch die Anweisung **uses** (engl. Anwendungen) mitteilen, daß auch die Unit CRT einzubauen ist. Wenn Sie die uses-Anweisung nicht in das Programm aufnehmen, reagiert der Compiler wie folgt:



2.2 Vereinbarungsteil

Auf den Programmkopf folgt in der Zeile 1 der Vereinbarungsteil (Deklarationsteil) des Programms. Im Vereinbarungsteil werden die Konstanten, Variablen oder sogar eigene Datentypen vereinbart. In welcher Reihenfolge die Vereinbarungen erfolgen, spielt in Turbo-Pascal keine Rolle.
Im Vereinbarungsteil wird der Computer angewiesen, Speicherplätze für die dort aufgeführten Variablen einzurichten. Variablen bezeichnen somit Speicherplätze im Hauptspeicher des Rechners, die mit Hilfe der angegebenen Variablennamen adressiert und wie-

dergefunden werden. Der Ausdruck Variable ist darauf zurückzuführen, daß ein Speicherplatz während der Ausführung eines Programms unterschiedliche, also variable Werte aufnehmen kann.
Die Variablenvereinbarung wird eingeleitet durch das Schlüsselwort **var**. Anschließend werden die Namen der Variablen angegeben. Die einzelnen Namen werden durch Komma getrennt. Leerzeichen zwischen einem Komma und einem Variablennamen haben keine Bedeutung. Sie werden vom Compiler überlesen.



Damit der Rechner weiß, wie groß der vorzusehende Speicherplatz sein muß, wird hinter den Variablennamen und einem Doppelpunkt der Datentyp angegeben.

Einfache Datentypen

In Turbo-Pascal kennt man folgende sogenannte einfache Datentypen:

(1) numerischer Datentyp

integer: ganzzahliger Wert
Beispiele: 21, –2355

real: reellwertige Zahl, zum Beispiel Zahlen mit Nachkommastellen
Beispiele: 34.17, –278.90

(2) Zeichen und Zeichenketten

char: Zeichen
Beispiele: A, M, 1, §

string: Zeichenkette

Beispiele: Computer, 15.11.1992
Hinweis: «15.11.1992» ist nicht numerisch, da die Angabe mehr als einen Punkt enthält.

(3) Wahrheitswert

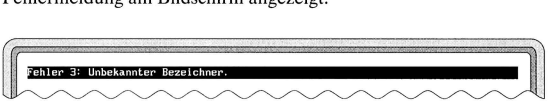
boolean: logische Variable, die entweder den Wahrheitswert wahr (true) oder falsch (false) aufnehmen kann.

Die Datentypen Integer, Char(acter) und Boolean werden auch als ordinale, abzählbare Datentypen bezeichnet. Bei ihnen ist nämlich die Reihenfolge eindeutig. Auf A folgt B, auf B folgt C. Der Nachfolger von 1 ist 2. Auf 2 folgt 3 usw.

Beispiele für Variablenvereinbarungen:

```
var Awert, Bwert: integer;
Steuer_Satz, Summe1, Summe2: real;
Kennsatz: char;
```

Das Schlüsselwort var braucht nur einmal angegeben zu werden, auch wenn sich die Variablenvereinbarung über mehrere Zeilen erstreckt. Die Namen von Variablen, die denselben Datentyp haben, können – durch Komma voneinander getrennt – aufgelistet werden. Das hat den Vorteil, daß die Datentypangabe nur einmal geschrieben werden muß. Hinter die Angabe des Datentyps ist ein Semikolon zu setzen!
In welcher Reihenfolge die Variablen vereinbart werden, ist für den Rechner unerheblich. Wichtig ist dagegen, daß jede Variable, die im Programm benutzt wird, vorher vereinbart worden ist. Sonst meldet das System einen Fehler.
Machen Sie die Probe! Löschen Sie in der Zeile 1 «Altkm,» aus der Variablenliste. Wenn Sie nun den Compiler starten, wird Ihnen eine Fehlermeldung am Bildschirm angezeigt.



Die Fehlermeldung besagt, daß ein im Programm benutzter Name (Bezeichner) nicht vorher vereinbart worden ist.
Wenn Sie nun «Altkm,» wieder in die Zeile 1 einfügen und kompilieren, so wird der Übersetzungslauf korrekt abgeschlossen.

Auch falsche Datentypangaben führen zu Fehlermeldungen. Ersetzen Sie in der Zeile 1 die Typangabe real durch integer.

```
var Altkm, Neukm, Liter, Verbrauch, Gefahrenekm: integer;
```

Bei der Kompilation wird die Fehlermeldung

angezeigt.
Wenn Sie Esc betätigen, steht der Cursor in der Zeile

```
Verbrauch := Liter / Gefahrenekm *100;
```

Das System prüft, ob der Datentyp des Ergebnisses einer Rechenoperation verträglich ist mit dem Datentyp der Variablen, in der sich das Ergebnis bildet. Der Datentyp des Ergebnisses bei einer Division ist stets real. Eine reellwertige Zahl belegt 6 Bytes. Für die Darstellung einer ganzzahligen Zahl werden dagegen nur 2 Bytes im Hauptspeicher bereitgestellt.
Da die Variable Verbrauch als ganzzahlige Variable vereinbart worden war, paßt somit das Ergebnis der Rechenoperation nicht in die Variable Verbrauch. Abgesehen von der ganzzahligen Division gilt:

> Die Variable, die das Ergebnis einer Division aufnehmen soll, muß vom Typ real sein.

2.3 Anweisungsteil

Auf den Vereinbarungsteil folgt der Anweisungsteil. Er wird durch ein **begin** eingeleitet und durch ein **end.** (mit Punkt!) abgeschlossen. Er beinhaltet die Anweisungen, die jeweils durch ein Semikolon voneinander zu trennen sind.
begin und **end** sind **keine** Anweisungen. Deshalb steht hinter ihnen kein Semikolon. Sie zeigen den Anfang und das Ende einer Folge von Anweisungen an.

```
begin
Anweisung;
Anweisung;
end.
```

Eingabeanweisung: READLN

Die Anweisung, mit der Daten über die Tastatur in den Rechner eingegeben werden, beginnt mit dem Schlüsselwort **readln**. Readln setzt sich zusammen aus read + line und bedeutet, daß der Cursor nach Abschluß der Eingabe auf den Anfang der nächsten Zeile gestellt wird.

```
readln (Altkm)
```

Hinter dem Schlüsselwort readln wird in runden Klammern der Name der Variablen angegeben, die den eingegebenen Wert aufnehmen soll. Man kann jedoch auch mehrere Variablennamen in einer Eingabeanweisung aufführen. So hätte man die Eingabe der Werte für Altkm und Neukm in einer Anweisung zusammenfassen können:

```
readln (Altkm, Neukm)
```

Die Variablennamen werden – durch Komma voneinander getrennt – zwischen den runden Klammern aufgelistet. Bei der Ausführung des Eingabebefehls wird der zuerst eingetastete Wert in die Variable Altkm, der nächste Eingabewert in die Variable Neukm gespeichert. Zwischen den beiden Eingaben ist die Leertaste zu betätigen. Die Leertaste erzeugt das erforderliche Trennzeichen zwischen den eingegebenen Zahlen.
In der Praxis hat es sich jedoch bewährt, in einer Eingabeanweisung immer nur einen Wert zu erfassen. Die Zusammenfassung von Eingaben in einer Eingabeanweisung spart Schreibarbeit. Bei der Ausführung des Programms jedoch ist sich der Benutzer des Programms unsicher, ob die Werte in der richtigen Weise vom Rechner übernommen worden sind. Deshalb empfiehlt es sich nicht, Eingaben in einer «Sammel»-Eingabeanweisung zu bündeln.

Eingabekorrekturen

Eine Eingabe kann, bevor sie durch ↵ an den Rechner «abgeschickt» wird, korrigiert werden. Haben Sie sich bei der Eingabe

eines Wertes vertippt, können Sie mit [←] das jeweils letzte Zeichen löschen.
Mit [Strg] [D] wird das zuletzt gelöschte Zeichen wiederhergestellt. [Esc] beziehungsweise [Strg] [X] löscht die gesamte Eingabe. Wurde die Eingabe insgesamt gelöscht, so kann sie mit [Strg] [R] wiederhergestellt werden.
[Strg] [Z] bewirkt den Abschluß einer Eingabe ohne Zeilenschaltung.

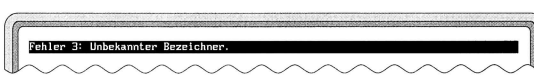
Ausgabeanweisung: WRITE / WRITELN

Das Schlüsselwort für die Ausgabeanweisung lautet **write** beziehungsweise **writeln**. Bei der Ausgabe über den Bildschirm bleibt bei Verwendung von **write** der Cursor hinter dem zuletzt ausgegebenen Zeichen stehen. Das Schlüsselwort **writeln** setzt sich aus write + line zusammen. Writeln bewirkt, daß nach der Ausgabe eines Wertes oder Textes eine Zeilenschaltung gemacht wird. Der Cursor steht dann am Anfang der nächsten Zeile.
Hinter dem Schlüsselwort write beziehungsweise writeln wird in runden Klammern aufgeführt, was ausgegeben werden soll.

```
write ('Wieviel Liter haben Sie getankt?');
```

Hier wird der Text «Wieviel Liter haben Sie getankt?» auf dem Bildschirm ausgegeben. Es handelt sich hier um eine Textkonstante (konstante Zeichenkette). Sie wird in Hochkommata gesetzt, um sie beispielsweise von Variablennamen unterscheiden zu können.
Haben Sie aus Versehen das abschließende Hochkomma vergessen, so erscheint beim Übersetzungslauf die Fehlermeldung:

Fehlt das einleitende Hochkomma, wird beim Kompilieren des Programms der Fehler



gemeldet.

Ist der Inhalt einer Variablen auszugeben, so wird der Name der Variablen in den runden Klammern angegeben.

Beispiel:

```
write (Verbrauch)
```

Die Ausgabeanweisung, mit der der Inhalt der Variablen Verbrauch angezeigt wird, ist in dem Programm VERBRAUC.PAS ein wenig aufwendiger gestaltet:

```
writeln ('Der Benzinverbrauch auf 100 km betrug ',Ver-
brauch:8:2,' Liter');
```

Die Zusammensetzung dieser Anweisung sollen Sie nun näher kennenlernen.
Lassen Sie sich das Programm VERBRAUC.PAS auf dem Bildschirm anzeigen. Fahren Sie den Cursor auf den Anfang der Zeile von

```
writeln ('Der Benzinverbrauch auf 100km betrug ',Ver-
brauch:8:2,' Liter');
```

und schaffen Sie sich durch Eingabe von Strg N eine Leerzeile. In diese schreiben Sie die Anweisung

```
writeln (Verbrauch);
```

Starten Sie das Programm, und geben Sie nun ein:

```
Wieviel Liter haben Sie getankt? 48 ↵
Kilometerstand beim vorletzten Tanken? 450 ↵
Kilometerstand beim letzten Tanken? 950 ↵
```

Am Bildschirm erscheint nun eine seltsame Zahl: 9.6000000000E+00. Es handelt sich um die Zahl 9.6, die in Gleitkommaschreibweise dargestellt ist. Die Darstellung besagt, daß die Zahl, die links vom E steht, positiv ist und mit dem Wert von $10^{00} = 1$ zu multiplizieren ist.

Beispiele für die Umrechnung von Gleitkommazahlen:

$5.00000000E\text{-}02 = 5 * 10^{-2} = 5 * 0.01 = 0.05$
$1.06000000E\text{+}01 = 1.06 * 10^{1} = 1.06 * 10 = 10.6$

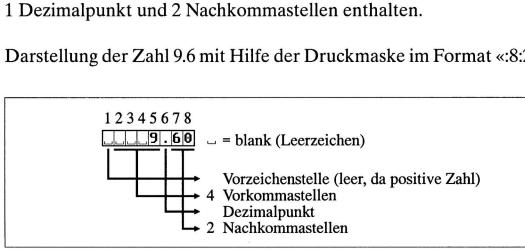
Ausgabeformatierung

Diese Form der Zahlendarstellung ist jedoch unübersichtlich. Deshalb gibt es die Möglichkeit, ein Ausgabeformat beziehungsweise eine Druckmaske anzugeben:

```
writeln (Verbrauch:8:2);
```

Die Angabe «:8:2» bedeutet, daß die Zahl mit insgesamt 8 Stellen ausgedruckt werden soll. In diesen 8 Stellen sind 1 Vorzeichenstelle, 1 Dezimalpunkt und 2 Nachkommastellen enthalten.

Darstellung der Zahl 9.6 mit Hilfe der Druckmaske im Format «:8:2»:



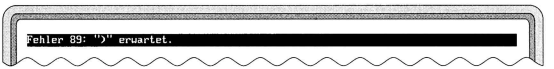
Bei der Ausgabe wird eine achtstellige Druckmaske gebildet, in die rechtsbündig die auszugebende Zahl eingetragen wird. Die links freibleibenden Stellen werden mit Leerzeichen aufgefüllt. Ein positives Vorzeichen (+) wird nicht ausgedruckt.
Ist die Druckmaske für die Vorkommastellen zu klein, so wird die Real-Zahl in Gleitkommaschreibweise ausgedruckt. Die Nachkommastellen werden so abgeschnitten, daß sie in die Ausgabemaske passen.
Auch bei der Ausgabe von ganzzahligen Werten kann die Angabe einer Druckmaske sinnvoll sein. Die Druckmaske für eine Integer-Variable unterscheidet sich von der Druckmaske einer Real-Variablen lediglich dadurch, daß die Angabe zu den Nachkommastellen fehlt. Wäre Verbrauch eine Integer-Variable, so hätte die Formatangabe in der Ausgabeanweisung

```
write (Verbrauch:8);
```

lauten können.
Damit der Computerbenutzer weiß, was eine ausgegebene Zahl bedeutet, kann man in die Ausgabeanweisung Textkonstanten zur Erläuterung einfügen. Konstante Texte werden, wie oben bereits erwähnt, in Hochkommata gesetzt. Die einzelnen Elemente einer Ausgabezeile werden durch Kommata voneinander getrennt.

```
                                1. Element: Textkonstante
writeln ('Der Benzinverbrauch auf 100km betrug ',
         Verbrauch:8:2,' Liter');

         2. Element:    3. Element:
         formatierte    Textkonstante
         Real-Zahl
```

Haben Sie vergessen, die einzelnen Elemente in der Druckanweisung durch Kommata voneinander zu trennen, meldet der Compiler den Fehler:



Diese Fehlermeldung scheint auf den ersten Blick unsinnig zu sein, da ja nur ein Komma fehlt. Der Compiler nimmt aber an, daß das nächste Element bereits zur folgenden Anweisung gehört. Er bricht deshalb den Übersetzungslauf ab, ohne festzustellen, daß einige Zeichen weiter die schließende runde Klammer steht.

Wertzuweisung: :=
Die Anweisung

```
Verbrauch := Liter *100 / Gefahrenekm;
```

ist eine Wertzuweisung. Das «:=»-Zeichen (Zuweisungsoperator) bedeutet «ergibt sich aus». Man kann die obige Anweisung wie folgt

lesen: Der Inhalt der Variablen Verbrauch ergibt sich aus dem Inhalt der Variablen Liter mal 100 geteilt durch den Wert in Gefahrenekm. Rechts von dem Zuweisungsoperator «:=» steht ein Ausdruck, der angibt, wie ein Wert gebildet wird. Links davon wird die Variable genannt, der der Wert zugewiesen wird. Der allgemeine Aufbau einer Wertzuweisung lautet somit:

```
Ergebnisvariable := Ausdruck
```

Beispiele:

```
MehrwertSteuersatz := 15;
Steuerbetrag := Warenwert * MehrwertSteuersatz / 100;
Rechnungsbetrag := Warenwert + Steuerbetrag;
```

Wie die Beispiele zeigen, können in dem Ausdruck Variablennamen und Zahlenkonstanten (beispielsweise 15) aufgeführt werden. Zahlenkonstanten werden nicht in Hochkommata gesetzt.
Folgende Rechenzeichen werden in Pascal verwendet:

+	Addition
-	Subtraktion
*	Multiplikation
/	Division

Ferner kennt Pascal zwei spezielle Formen der Division:
(1) Ganzzahlige Division

```
Wert := 108 div 5
```
In Wert bildet sich die Zahl 21.

(2) Divisionsrest-Berechnung

```
Wert := 108 mod 5
```
In Wert bildet sich die Zahl 3.

Bei Rechenoperationen gilt die in der Mathematik übliche Regel:

Punktrechnung vor Strichrechnung

Die Reihenfolge der Berechnung kann jedoch durch Klammern geändert werden. Geklammerte Ausdrücke werden zuerst bearbeitet.

Beispiel:

```
x := 5 * 2 + 7     x enthält den Wert 17
x := 5 * (2 + 7)   x enthält den Wert 45
```

Standardprozeduren

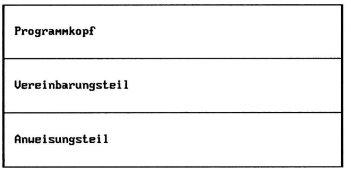
ClrScr ist eine «Standardprozedur» und bedeutet Clear Screen = Lösche den Bildschirm. Dabei wird der Bildschirm gelöscht und der Cursor in die Home-Position, das heißt in die linke obere Bildschirmecke gebracht.

Standardprozeduren erleichtern das Programmieren. Sie stellen kleine Programme dar, die vom Pascal-System zur Verfügung gestellt werden. Der Aufruf erfolgt über den Prozedurnamen.

Turbo-Pascal verfügt über eine große Anzahl von Standardprozeduren und Standardfunktionen. Auf sie wird im Kapitel 7 eingegangen.

2.4 Zusammenfassung

- Ein Pascal-Programm setzt sich in der Regel aus drei Teilen zusammen:



- Ein Name beginnt mit einem Buchstaben. Außer Buchstaben, Ziffern und Unterstreichern dürfen in einem Namen keine anderen Zeichen verwendet werden.
- Ein Name darf höchstens 127 Stellen lang sein.

- Eine Programmzeile darf 127 Zeichen umfassen.
- Das Semikolon trennt zwei Anweisungen voneinander.
- Jedes Sprachelement ist durch ein Begrenzungszeichen von dem folgenden zu trennen. Begrenzungszeichen sind:
  Leerzeichen
  Zeilenende
  Kommentar
- Eine Text-Konstante wird in Hochkommata gesetzt.
- In Turbo-Pascal gibt es folgende vordefinierte Konstanten:

Name	Wert	Datentyp
maxint	32767	integer
pi	3.1415926536E+00	real
true	Wahrheitswert richtig	boolean
false	Wahrheitswert falsch	boolean

- Das Schlüsselwort für die Eingabeanweisung lautet readln.
- Das Schlüsselwort für die Ausgabeanweisung lautet write beziehungsweise writeln.
- := bezeichnet man als Zuweisungsoperator.
- Tabelle der vordefinierten ganzzahligen Datentypen:

Typname	Wertebereich		Format
Byte	0 ...	255	1 Byte
Word	0 ...	65535	2 Bytes
ShortInt	-128 ...	127	1 Byte
Integer	-32768 ...	32767	2 Bytes
LongInt	-2147483648 ...	2147483647	4 Bytes

- Tabelle der vordefinierten reellwertigen Datentypen:

Typname	Wertebereich			Format
Single	$1.5 * 10^{-45}$	...	$3.4 * 10^{38}$	6 Bytes
Real	$2.9 * 10^{-39}$	...	$1.7 * 10^{38}$	4 Bytes
Double	$5.0 * 10^{-324}$	...	$1.7 * 10^{308}$	8 Bytes
Extended	$1.9 * 10^{-4951}$	...	$1.1 * 10^{4932}$	10 Bytes
Comp	$-9.2 * 10^{18}$	...	$9.2 * 10^{18}$	8 Bytes

Tabelle der Units

Unit-Name	Aufgabengebiet
System	Enthält die Grundfunktionen, die für jedes lauffähige Programm benötigt werden. Die Unit System wird vom Compiler automatisch geladen und wird somit nicht in einer uses-Anweisung aufgeführt.
Crt	Bildschirmsteuerung (beispielsweise ClrScr, Gotoxy, ClrEol), Farben, Fenster, Tastaturkontrolle, Ton
Printer	Ausgabe von Daten über den Drucker
DOS	Nutzung von DOS-Funktionen: Aufruf von Daten und Uhrzeit, Suchen in Verzeichnissen, Interrupts und Grafikroutinen
Graph	Grafikanwendungen
Overlay	Verwaltung von Overlays
Turbo3	Anpassungsroutinen zu Turbo-Pascal Version 3
Graph3	Anpassungsroutinen zur Grafik der Version 3

Die Namen der Units, die in das Programm einzubinden sind, werden hinter dem Schlüsselwort uses aufgeführt und durch Komma voneinander abgetrennt.

2.5 Übungen

1. Welche Fehler enthält die Formulierung des Programmkopfes

   programm Kilometer-Berechnung;

   a) _____
   b) _____

2. Warum müssen Variablen, mit denen im Anweisungsteil gearbeitet werden soll, vorher im Vereinbarungsteil mit Angabe des Datentyps vereinbart werden?
3. Welche Bedeutung hat das Semikolon in Pascal?
4. Das Programm Zinsrechnung weist einige syntaktische Fehler auf. Korrigieren Sie die Fehler. Geben Sie das Programm ein, und testen Sie es.

Anmerkung: Die Zeilennummern sind nicht Bestandteile des Programms. Sie dienen lediglich der besseren Orientierung. Sie dürfen also nicht eingegeben werden.

```
 0  program Zinsrechnung;
    uses crt;                                   (* ab Version 4 *)
 1  var Zinsen, Betrag: real;
 2      Endkapital: integer;
 3  begin
 4    ClrScr
 5    write ('Eingabe Betrag: ');
 6    readln (Betrag);
 7    wirte ('Eingabe Zinssatz: ');
 8    readln (Zinssatz);
 9    Zinsen := Betrag * 0.01 * Zinssatz/ 360;
10    Endkapital := Betrag + Zinsen;
11    writeln ('Zinsen:', Zinsen:10);
12    writeln ('Endbetrag:', Endkapital:10:2);
13  end.
```

2.6 Lösungen

1. a) Das Schlüsselwort program wird nur mit einem m geschrieben.
   b) Als Sonderzeichen ist lediglich der Unterstreicher zugelassen.
2. Durch die Variablenvereinbarung wird ein Speicherplatz bestimmter Größe mit einem Namen versehen. Unter diesem Namen wird der Speicherplatz angesprochen. Dessen Größe wird durch die Angabe des Datentyps festgelegt.
3. Durch das Semikolon wird eine Anweisung von der folgenden Anweisung getrennt.
4. Syntaktische Fehler des Programms Zinsrechnung:
   Zeile 4:
   Hinter ClrScr fehlt das Semikolon.
   Zeile 7:
   Das Schlüsselwort für die Ausgabeanweisung lautet **write**.
   Zeile 8:
   Die Variable Zinssatz wurde nicht definiert. Der Fehler wird behoben, indem in Zeile 1 hinter Betrag **«, Zinssatz»** eingefügt wird.
   Zeile 10:
   Endkapital wurde als Integer-Variable vereinbart. Da sich der

Wert für Endkapital ergibt aus der Addition von zwei Real-Variablen, muß die Ergebnisvariable Endkapital ebenfalls vom Typ real sein. Fehlerbeseitigung: Ersetzen Sie in der Zeile 2 «integer» durch «real».

Zeile 11:

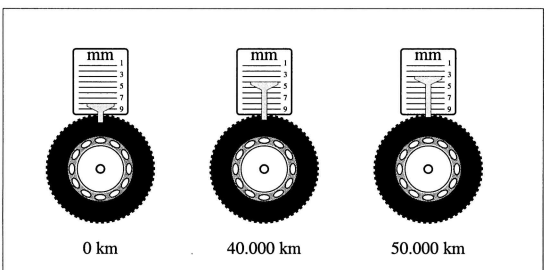
Zinsen ist eine Real-Variable. Deshalb ist die Druckmaske falsch. Korrekte Druckmaske: **Zinsen:10:2**.

3 ENTWICKLUNG EINES PROGRAMMS

Ob ein Programm funktioniert, hängt nicht von der Sprache ab, in der es geschrieben ist. Sicherlich ist Turbo-Pascal eine Programmiersprache, in der man sehr gut programmieren kann. Ob aber ein Programm die gewünschte Leistung erbringt, ist in erster Linie vom richtigen Aufbau des Lösungsweges zu einem gegebenen Problem abhängig. Deshalb soll in diesem Kapitel dargestellt werden, wie man ein Programm entwickelt.

***Programmbeispiel* Reifenabnutzung**

Zwei Kollegen unterhalten sich in der Frühstückspause. «Hast du in der Zeitung schon das Reifenangebot vom Reifenhändler Fritz gelesen?» «Nein, zeig mal her!» – Er studiert die Anzeige. «Mensch, das sind ja wirklich tolle Preise. Ob es sich lohnt, sich jetzt schon neue Reifen zuzulegen?» «Wieso, du hast in deiner Abteilung ja einen Computer stehen. Laß es dir doch von ihm ausrechnen.»



3.1 Klärung der Problemstellung: Problemanalyse

Das Pausengespräch endete mit dem Vorschlag, der Computer solle ausrechnen, ob sich der sofortige Reifenkauf lohnt. Nur, wann lohnt er sich? Wahrscheinlich wird der sofortige Kauf günstig sein, wenn das Auto ohnehin in nur wenigen Wochen mit neuen Reifen zu versehen ist, weil die alten «Galoschen» bereits stark abgefahren sind. Die Überlegungen laufen also auf die Frage hinaus: «Wie viele Kilometer kann man mit den alten Reifen noch ungefähr fahren?»
Damit ist die Problemstellung schon teilweise geklärt. Der Computer soll ausrechnen, wieviele Kilometer noch mit den alten Reifen gefahren werden können. Dabei ist zu berücksichtigen, daß nach Meinung der Experten die Reifen mindestens 2 mm Profiltiefe haben müssen, da sie sonst nicht mehr verkehrssicher sind.
Daran schließt sich die Frage an, wie groß der Reifenverschleiß im Durchschnitt pro Kilometer ist. Der durchschnittliche Reifenverschleiß läßt sich aus der bisherigen Reifenabnutzung und den gefahrenen Kilometern berechnen.

Berechnungsbeispiel: Ein neuer Reifen hat eine Profiltiefe von 8 mm. Nach 25000 km wird eine Profiltiefe von 3 mm gemessen. Der Reifenverschleiß errechnet sich aus

anfängliche Profiltiefe	8 mm
– gemessenes Profil	3 mm
Reifenverschleiß	5 mm

Der durchschnittliche Reifenverschleiß pro Kilometer beträgt dann:

$$\frac{5}{25000} = 0{,}0002 \text{ mm Reifenverschleiß pro Kilometer}$$

Das noch nutzbare Profil bestimmt sich aus der Differenz von gemessener Profiltiefe und dem Sicherheitsprofil von 2 mm. So läßt sich leicht berechnen, nach wieviel Kilometern das Reifenprofil von 3 mm auf 2 mm abgefahren sein wird:

$$\frac{1}{0{,}0002} = 5000 \text{ km}$$

Die Reifen können also noch weitere 5000 km laufen. Dabei wurde eine gleichmäßige Abnutzung pro Kilometer unterstellt.
Damit ist das Problem, das der Computer lösen soll, geklärt. Bevor die Problemlösung einem Computer übertragen wird, muß festgelegt werden,

was der Computer berechnen soll
und
wie die Berechnung erfolgen soll.

3.2 Datenanalyse

In dem zweiten Arbeitsschritt «Datenanalyse» wird festgelegt

welche Daten auszugeben sind
und
welche Daten einzugeben sind.

Ferner ist zu untersuchen, über welche Zwischenschritte die Ausgabedaten aus den Eingabedaten gewonnen werden. Schließlich ist zu ermitteln, von welchem Datentyp die Daten sind.

Festlegung der Ausgabedaten
Der Rechner soll ausgeben, wie viele Kilometer die Reifen noch laufen können. Die Variable, in der sich dieser Wert bilden soll, sei LaufKm genannt. Wie in der Problemanalyse dargestellt wurde, ergibt sich dieser Wert aus einer Division. Also muß die Ausgabevariable vom Typ Real sein.
Als Ausgabedatum wird festgelegt:

```
LaufKm      :Real
```

Festlegung der Eingabedaten
Auf Grund der Problemanalyse ist bekannt, welche Daten einzugeben sind. Zum einen ist anzugeben, wie viele Kilometer mit den

Reifen bereits gefahren wurden. Zum anderen ist dem Rechner mitzuteilen, wieviel Profil die Reifen noch aufweisen. Die Variable, in die die Anzahl der gefahrenen Kilometer einzugeben ist, wird GefahreneKm genannt. Der Wert für die Profiltiefe soll in der Variablen mit dem Namen GemessenesProfil gespeichert werden.
Nun ist zu überlegen, von welchem Datentyp diese Variablen sein müssen. Die Angabe zu dem gemessenen Profil erfolgt in Millimeter. Sie kann in einer ganzzahligen Variablen gespeichert werden.
Bei der Kilometerangabe wird in der Regel auch nur eine ganze Zahl erwartet, zum Beispiel 25000 oder 40000 Kilometer. In einer ganzzahligen Variablen können jedoch nur Werte bis 32767 gespeichert werden. Da aber die Anzahl der zurückgelegten Kilometer diesen Wert übersteigen könnte, muß die Variable GefahreneKm als Real-Variable vereinbart werden. Ab der Version 4 können Sie auch den Datentyp LongInt verwenden.
Somit kann man als Eingabedaten festlegen:

```
GefahreneKm      :    Real
GemessenesProfil :    Integer
```

Beziehungen zwischen Ein- und Ausgabedaten
Nun ist zu untersuchen, wie die Ausgabedaten aus den Eingabedaten gewonnen werden. Aus der Problemanalyse ging hervor, daß die noch zu fahrenden Kilometer berechnet werden aus der Formel

$$\text{LaufKm} = \frac{\text{NutzbaresProfil}}{\text{DurchschnittVerschleiss}}$$

Der Wert für NutzbaresProfil wird ermittelt durch

Gemessenes Profil – Sicherheitsprofil

Damit ist der Wert für NutzbaresProfil auf einen Eingabewert und eine Konstante zurückgeführt.
Der Wert DurchschnittVerschleiss errechnet sich aus

$$\frac{\text{ReifenVerschleiss}}{\text{GefahreneKm}}$$

Der Wert für GefahreneKm ist ein Eingabewert. Der Wert für ReifenVerschleiss berechnet sich aus

Anfangsprofil – GemessenesProfil

Anfangsprofil ist mit 8 (mm) ein konstanter Wert. Bei dem Wert für GemessenesProfil handelt es sich um einen Eingabewert.
Somit ist die Kette der Zwischenschritte geschlossen. Man stellt sie in einer Pascal-ähnlichen Schreibweise zusammen.

```
LaufKm := NutzbaresProfil / DurchschnittVerschleiss
NutzbaresProfil := GemessenesProfil – Sicherheitsprofil
DurchschnittVerschleiss := ReifenVerschleiss / GefahreneKm
ReifenVerschleiss := Anfangsprofil – GemessenesProfil
```

Der Typ der Variablen NutzbaresProfil, ReifenVerschleiss und DurchschnittVerschleiss ist noch festzulegen.
Bei der Festlegung der Datentypen von Variablen kann man folgende Faustregel anwenden:
Eine Ergebnisvariable muß vom Typ Real sein,
(1) wenn zur Berechnung ihres Wertes Real-Variablen herangezogen werden oder
(2) wenn sich ihr Wert durch eine Division ergibt oder
(3) wenn ihr Wert größer als 32767 oder kleiner als –32768 sein kann.

Da sich der Wert für DurchschnittVerschleiss durch eine Division ergibt, muß sie vom Typ Real sein. Die Werte der Variablen ReifenVerschleiss und NutzbaresProfil ergeben sich jeweils als Differenz zweier ganzzahliger Werte. Somit kann der Datentyp dieser Variablen vom Typ Integer sein.
Zusammenstellung der Variablen:

```
GemessenesProfil, ReifenVerschleiss, NutzbaresProfil: Integer
GefahreneKm, LaufKm, DurchschnittVerschleiss: Real
```

Als Konstanten sind festgelegt worden:

```
Anfangsprofil = 8
Sicherheitsprofil = 2
```

Damit ist die Datenanalyse abgeschlossen.

Anmerkung zur Schreibweise von Variablennamen: In Variablennamen sollten die Buchstaben ß, ä, ü und ö nicht verwendet werden. Sie haben im ASCII-Zeichensatz eine andere Bedeutung.

3.3 Aufbau des Lösungsweges (Algorithmus)

Im Lösungsweg (Algorithmus) wird festgelegt, in welcher Reihenfolge die einzelnen Arbeitsschritte vom Computer ausgeführt werden müssen, um zum richtigen Ergebnis zu kommen. **Als Algorithmus bezeichnet man eine Folge von Anweisungen (Arbeitsschritten) zur Lösung eines Problems.** Zur Darstellung des Algorithmus kann man folgende Methoden anwenden:

- Pseudocode
- Programmablaufplan nach DIN 66001
- Struktogramm nach DIN 66261

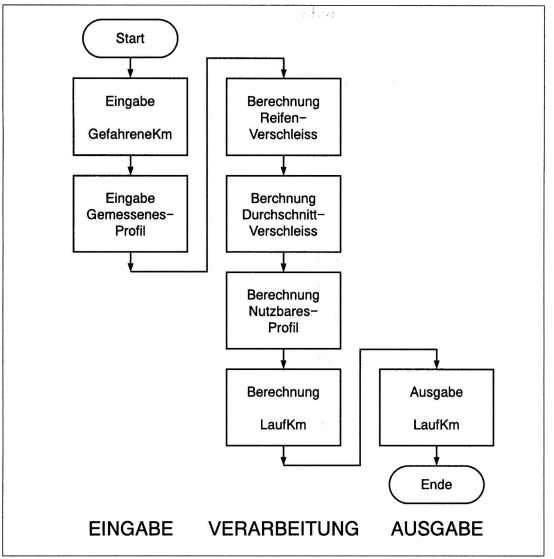
Die verschiedenen Darstellungsmethoden werden in diesem Buch teilweise nebeneinander verwendet. Der Leser kann dann letztlich die Darstellungsweise für sich auswählen, die ihm am meisten zusagt. Die Symbole für den Programmablaufplan und für das Struktogramm sind im Anhang dargestellt. Der Pseudocode ist dadurch gekennzeichnet, daß er ziemlich frei formuliert werden kann. Häufig bedient man sich aber einer Schreibweise, die sich der Pascal-Sprache schon stark annähert. Die Darstellung des Algorithmus zum Reifenproblem im Pseudocode könnte wie folgt lauten:

```
Algorithmus  Reifenabnutzung
Konstanten   Anfangsprofil = 8
             Sicherheitsprofil = 2
Beginn
   Eingabe der gefahrenen Kilometer
   Eingabe des gemessenen Reifenprofils
   Berechne den durchschnittlichen Verschleiss
   Berechne die Kilometer, die die Reifen noch genutzt
   werden können
   Ausgabe der Kilometerzahl
Ende
```

Diesen Algorithmus, der nahezu umgangssprachlich formuliert ist, kann man noch genauer beschreiben, indem man die oben festgelegten Variablennamen und Formeln verwendet. Der Algorithmus lautet dann in der verfeinerten Fassung:

```
Algorithmus  Reifenabnutzung
Konstante    Anfangsprofil = 8
             Sicherheitsprofil = 2
Beginn
   Eingabe GefahreneKm
   Eingabe GemessenesProfil
   ReifenVerschleiss := Anfangsprofil – GemessenesProfil
   DurchschnittVerschleiss := ReifenVerschleiss / GefahreneKm
   NutzbaresProfil := GemessenesProfil – Sicherheitsprofil
   LaufKm := NutzbaresProfil / DurchschnittVerschleiss
   Ausgabe der LaufKm
Ende
```

In Form eines Programmablaufplans wird der Algorithmus wie folgt dargestellt:



Als Struktogramm hat der Algorithmus folgende Form:

Algorithmus Reifenabnutzung
Konstante Anfangsprofil = 8 Sicherheitsprofil = 2
Eingabe GefahreneKm
Eingabe GemessenesProfil
ReifenVerschleiss := Anfangsprofil - Gemessenes Profil
DurchschnittVerschleiss := ReifenVerschleiss / GefahreneKm
NutzbaresProfil := GemessenesProfil - Sicherheitsprofil
LaufKm := NutzbaresProfil / DurchschnittVerschleiss
Ausgabe LaufKm
Ende

3.4 Erstellung des Programms

Nachdem der Datenrahmen abgeklärt und der Lösungsweg festgelegt ist, kann nun das Programm erstellt werden. Dieser 4. Schritt besteht darin, die in den Schritten 2 und 3 getroffenen Festlegungen bezüglich Variablennamen, Datentypen und Reihenfolge der Anweisungen in eine Programmiersprache zu übertragen. Dabei sind die Syntaxregeln der Programmiersprache zu beachten. Die für die Lösung dieses Problems notwendigen Syntaxregeln wurden im zweiten Kapitel bereits weitgehend behandelt. Lediglich die Vereinbarung von Konstanten ist noch zu klären.

Konstantenvereinbarung
Eine Konstante ist eine Größe mit einem bestimmten Wert, der sich während der Laufzeit des Programms nicht verändert. Im Vereinbarungsteil des Programms kann man ihr einen Namen zuordnen, der im Programm an Stelle des konstanten Wertes verwendet werden kann.
Die Konstantenvereinbarung lautet allgemein

```
const Name = Wert;
```

Die Vereinbarung von Konstanten wird eingeleitet durch das Schlüsselwort **const**. Darauf folgt nach einer Leerstelle der Name der Konstanten, der mit einem bestimmten Wert gleichgesetzt wird. Ein Semikolon schließt die Vereinbarung ab.

Beispiele:

```
const   Anfangsprofil = 8;
        Sicherheitsprofil = 2;
        Ueberschrift = 'Reifenproblem';
```

Wie aus den Beispielen zu ersehen ist, kann nicht nur eine konstante Zahl, sondern auch eine Textkonstante einem Namen zugeordnet werden. Das Schlüsselwort const braucht nur einmal geschrieben zu werden.
Die Verwendung von Namen für Konstanten ist in zweierlei Hinsicht vorteilhaft:

(1) Die Programme sind leichter lesbar.
(2) Wenn Konstanten geändert werden müssen, braucht die Änderung nur im Vereinbarungsteil des Programms vorgenommen werden.

Somit läßt sich das Pascal-Programm zu dem Problem Reifenabnutzung wie folgt schreiben:

```
program Reifenabnutzung;
uses  crt;                                          (* ab Version 4 *)
const Anfangsprofil = 8;
      Sicherheitsprofil = 2;
      Ueberschrift = 'Reifenproblem';
var GemessenesProfil, ReifenVerschleiss, NutzbaresProfil: integer;
    GefahreneKm, DurchschnittVerschleiss, LaufKm : real;

begin
  ClrScr;
  writeln (Ueberschrift);
  write ('Gefahrene Kilometer? '); readln (GefahreneKm);
  write ('Gemessenes Profil in mm? '); readln (GemessenesProfil);
  ReifenVerschleiss := Anfangsprofil - GemessenesProfil;
  DurchschnittVerschleiss := Reifenverschleiss / GefahreneKm;
  NutzbaresProfil := GemessenesProfil - Sicherheitsprofil;
  LaufKm := NutzbaresProfil / DurchschnittVerschleiss;
  writeln('Sie können noch ',LaufKm:8:2,' km mit den Reifen fahren');
end.
```

Mit der Umsetzung vom Algorithmus in ein Programm kann man erst dann beginnen, wenn folgende Fragen beantwortet sind:

- Welche Variablen werden benötigt?
- Welchen Datentyp haben die Variablen?
- Wie sind die Werte zu bilden?
- Welche Reihenfolge der Arbeitsschritte muß eingehalten werden?

Dies wird mit Absicht betont, weil eine unsystematische Programmentwicklung zu Mißerfolgen führt!

3.5 Eingabe des Programms

Das oben formulierte Programm wird im nächsten Arbeitsschritt eingegeben. Die Eingabeweise ist im ersten Kapitel bereits beschrieben worden.

Folge der Arbeitsschritte	Kommando		
	Version 3	Version 4, 5, 6	Version 7
Aufruf des Turbo-Pascal-Systems	turbo ↵	turbo ↵	turbo ↵
Include error messages?	Y (es)		
Umstellen auf das aktuelle Laufwerk (LW)	L Eingabe LW: ↵	F10 F C Eingabe LW:	F10 D V Eingabe
und Verzeichnis	A Eingabe Verzeichnis ↵	\Verzeichnis ↵	LW:\Verzeichnis ↵
Vereinbaren der Arbeitsdatei	W Eingabe Dateiname ↵		
Aufruf des Editors	E	F10 F	F10 D
Eingabe des Programms		N	N
Abschluß der Editierarbeit	Strg K D Leertaste		
Speichern des Programms	S	F2 Eingabe Dateiname ↵	F2 Eingabe Dateiname ↵

3.6 Programmtest

3.6.1 Syntaxtest

Der Programmtest wird in zwei Phasen vollzogen. In der ersten Phase wird geprüft, ob das Programm syntaktisch richtig ist. Hier wird also untersucht, ob das Programm den Sprachregeln der Pascal-Sprache entspricht. Rufen Sie aus dem Hauptmenü durch Eingabe von C den Compiler auf. Der Compiler überprüft beim Übersetzen des eingegebenen Quellprogramms in das Maschinenprogramm, ob das Quellprogramm den Syntaxregeln entspricht.

Häufige Syntaxfehler:

- Hinter einer Anweisung fehlt das Semikolon.
- Ein Variablenname wurde nicht im Deklarationsteil vereinbart bzw. ein Variablenname wurde im Anweisungsteil anders geschrieben als vorher vereinbart.
- Ein Schlüsselwort wurde falsch geschrieben, zum Beispiel wirteln statt writeln.
- Eine Textkonstante wurde nicht in zwei Hochkommata gesetzt.
- Einer Variablen wurde ein falscher Datentyp zugeordnet.

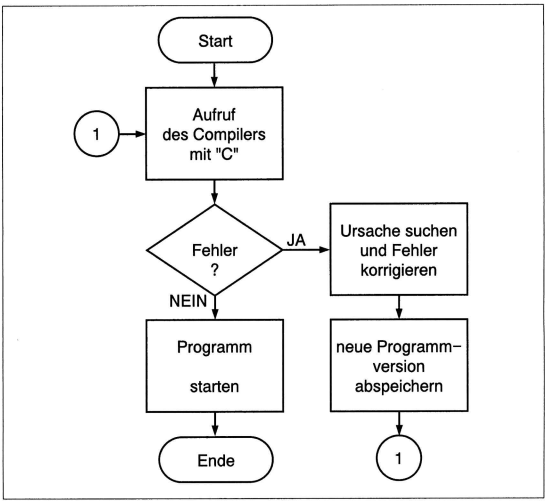
Stellt der Compiler einen Syntaxfehler fest, so bricht er den Übersetzungslauf ab und gibt an, in welcher Programmzeile welcher Fehler aufgetreten ist. Nach Betätigen der Taste [Esc] kehrt das System in den Editor zurück. Das Programm wird auf dem Bildschirm angezeigt. Der Cursor steht an der Stelle, an der der Fehler festgestellt wurde.
Obwohl der Cursor an der Stelle steht, wo der Fehler aufgetreten ist, kann es einige Mühe bereiten, die Fehlerursache im Programm auszumachen. Der eigentliche Fehler kann nämlich einige Programmzeilen weiter oben liegen. Beispielsweise hat der Compiler festgestellt, daß ein Variablenname unbekannt ist. Hier können jetzt zwei Fehlerursachen vorliegen:

(1) Bei der Eingabe der Variablenbezeichnung hat man sich vertippt. Statt GemessenesProfil hat man beispielsweise GemesenesProfil eingegeben.
(2) Es wurde vergessen, die Variablenbezeichnung im Deklarationsteil zu vereinbaren.

Wird also ein Fehler aufgezeigt, so müssen Sie kritisch prüfen, worin die eigentliche Fehlerursache liegt!
Nachdem Sie den Fehler korrigiert haben, speichern Sie das korrigierte Programm und rufen den Compiler wieder auf.

Version 3	ab Version 4
[Strg] [K] [D]	[F2]
[S]	[F10] [C]
[C]	[C]

Das Ablaufdiagramm für die Fehlersuche hat die folgende Form:



Zu weiteren Einzelheiten der Übersetzungsphase siehe Kapitel 1.

3.6.2 Logischer Test

Wurde das Programm ordnungsgemäß übersetzt, können Sie nun mit der zweiten Testphase beginnen. In dieser Testphase wird das Programm daraufhin geprüft, ob das Ergebnis des Programmlaufs korrekt ist. In erster Linie geht es darum festzustellen, ob der Programmablauf dem logischen Ablauf der Problemlösung entspricht.
Logische Fehler können gefährlich sein. Sie geben Daten ein, der Rechner zeigt Ihnen Ergebnisse an, und Sie könnten nun glauben, daß die Computerergebnisse richtig sind, weil der Computer sie errechnet hat.
Aber: **Glauben ist gut, Kontrolle ist besser!**

Um die Gefährlichkeit logischer Fehler zu demonstrieren, sollten Sie folgende Programmveränderung vornehmen. Vertauschen Sie die beiden Zeilen

```
NutzbaresProfil := GemessenesProfil – Sicherheitsprofil;
LaufKm := NutzbaresProfil / DurchschnittVerschleiss;
```

miteinander. Dazu setzen Sie den Cursor auf den Anfang der Zeile

```
NutzbaresProfil := Gemessenes Profil – Sicherheitsprofil;
```

und geben die Tastenkombination Strg K B (= Markieren Blockanfang) ein. Anschließend fahren Sie den Cursor hinter das Semikolon und markieren mittels Strg K K das Ende des Blocks.
Nun stellen Sie den Cursor auf den Anfang der Druckanweisung

```
writeln ('Sie können noch ',LaufKm:8:2,' km mit den
Reifen fahren');
```

und geben die Tastenkombination Strg K V ein. Die vorher markierte Zeile erscheint nun nach der Zeil

```
LaufKm := NutzbaresProfil / DurchschnittVerschleiss;
```

Mit Strg K H heben Sie die Blockmarkierung wieder auf.
Das Programm lautet somit:

```
program Reifenabnutzung;
uses  crt;                                    (* ab Version 4 *)
const Anfangsprofil = 8;
      Sicherheitsprofil = 2;
      Ueberschrift = 'Reifenproblem';
var GemessenesProfil, ReifenVerschleiss, NutzbaresProfil: integer;
    GefahreneKm,DurchschnittVerschleiss, LaufKm : real;

begin
  ClrScr;
  writeln (Ueberschrift);
  write ('Gefahrene Kilometer? '); readln (GefahreneKm);
  write ('Gemessenes Profil in mm? '); readln (GemessenesProfil);
  ReifenVerschleiss := Anfangsprofil - GemessenesProfil;
  DurchschnittVerschleiss := Reifenverschleiss / GefahreneKm;
  LaufKm := NutzbaresProfil / DurchschnittVerschleiss;
  NutzbaresProfil := GemessenesProfil - Sicherheitsprofil;
  writeln('Sie können noch ',LaufKm:8:2,' km mit den Reifen fahren');
end.
```

Wenn Sie nun das Programm übersetzen, wird kein Fehler festgestellt, da das Programm syntaktisch in Ordnung ist. Starten Sie das Programm, so erleben Sie einige Überraschungen.
Geben Sie ein:

```
Gefahrene Kilometer? 25000 ↵
Gemessenes Profil? 3 ↵
```

Der Computer zeigt als Ergebnis an:

```
Sie können noch 15000.00 km mit den Reifen fahren
```

Möglicherweise wird auch ein anderes Ergebnis angezeigt. Der Grund dafür liegt darin, daß die Variable NutzbaresProfil bei der Berechnung von LaufKm herangezogen wird, obwohl ihr Wert noch nicht bestimmt worden ist. Das geschieht erst in der folgenden Anweisung. Auf jeden Fall ist das Ergebnis falsch, obwohl es korrekt wirkt. Das zeigt folgende Rechnung:

```
ReifenVerschleiss = 8 – 3 = 5
DurchschnittVerschleiss = 5 / 25000 = 0.0002
NutzbaresProfil = 3 – 2 = 1
LaufKm = 1 / 0.0002 = 5000
```

Der Vergleich der beiden Ergebnisse zeigt, daß das Computerergebnis falsch ist. Sie können noch weitere Kontrollrechnungen durchführen und dabei stets feststellen, daß die «manuellen» Ergebnisse vom Computerergebnis abweichen.

Zur Überprüfung der logischen Richtigkeit eines Programms sind Zahlenbeispiele «manuell» durchzurechnen. Anschließend werden die Zahlen der Beispiele in den Rechner eingegeben. Stimmen jeweils die Ergebnisse überein, kann man im allgemeinen davon ausgehen, daß das Programm richtig ist.

3.6.3 Programm schrittweise ausführen (Trace)

Die Suche nach logischen Fehlern kann sehr mühselig und zeitraubend sein. Deshalb wurde ein **Debugger** (engl. Entlauser) in das Tur-

bo-Pascal-System integriert, der bei der Fehlersuche sehr hilfreich ist. Er ermöglicht es Ihnen, ein Programm schrittweise ausführen zu lassen. Da hierbei Ihr Quellprogramm sichtbar ist, können Sie erkennen, welche Anweisung gerade bearbeitet wird. Diese schrittweise Programmausführung wird auch **Trace** (engl. eine Spur verfolgen) genannt. Ferner gestattet Ihnen die integrierte Entwicklungsumgebung, sich den Inhalt von Variablen anzeigen zu lassen. Schließlich können Sie sogar während des Trace-Laufs «von außen» Variableninhalte ändern.

Voraussetzungen

Die schrittweise Programmausführung setzt zum einen voraus, daß das Programm syntaktisch fehlerfrei ist. Zum anderen müssen bestimmte Parameter gesetzt sein, damit das System beim Kompilieren des Programms die erforderliche Testumgebung schaffen kann.

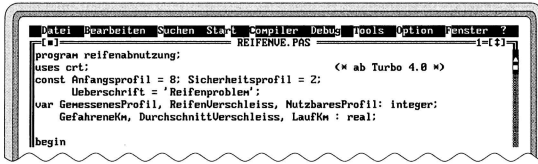
Sie lauten in den verschiedenen Versionen:

Version 7	Version 6	Version 5.x
Option/Compiler	**Option/Compiler**	**Options/Compiler**
[X] Debug-Informationen	[X] Debug information	Debug information on
[X] Lokale Symbole	[X] Local symbols	Local symbols on
Option/Debugger	**Option/Debugger**	**Debug**
[X] Integrierter Debugger	[X] Integrated	Integrated debugging on

Die Vorgehensweise bei der schrittweisen Programmausführung sei am Beispiel des verfälschten Programms Reifenabnutzung aufgezeigt.

Programmablauf verfolgen (Tracen)

Sie starten den Trace-Lauf mittels F7. Auf dem Beginn des Anweisungsteils erscheint ein Balken, hier Tracebalken genannt. Ein erneutes Betätigen von F7 stellt den Balken in die nächste Zeile; er steht nun auf der Anweisung ClrScr. Der Tracebalken gibt an, welche Anweisung im nächsten Schritt ausgeführt wird.



Wenn Sie nun F7 anschlagen, ist die Anweisung in dieser Zeile ausgeführt. Überzeugen Sie sich: Schalten Sie mittels Alt F5 in den Ausgabebildschirm um. Er ist gelöscht! Durch Betätigen einer beliebigen Taste schalten Sie wieder in den Editbildschirm um.

> Durch Betätigen von F7 wird die Anweisung, die sich unter dem Tracebalken befindet, ausgeführt.

Die Befehlsausführung erfolgt zeilenweise. Stehen mehrere Anweisungen in einer Zeile, werden sie ohne «Zwischenstop» ausgeführt. Wenn Sie die Reihenfolge der Befehlsbearbeitung beobachten wollen, sollten Sie die den Programmablauf steuernden Anweisungen nicht in einer Zeile zusammenfassen.

Beispiel:

NICHT:

```
if Altkm < Neukm then Gefahrenekm := Neukm + 10000 - Altkm else Gefahrenekm :=
Neukm - Altkm;
```

sondern:

```
if Altkm < Neukm
   then Gefahrenekm := Neukm + 10000 - Altkm
   else Gefahrenekm := Neukm - Altkm;
```

Bei der zweiten Form der Schreibweise können Sie beim Tracen genau nachvollziehen, ob die THEN-Anweisung oder die ELSE-Anweisung ausgeführt wurde.

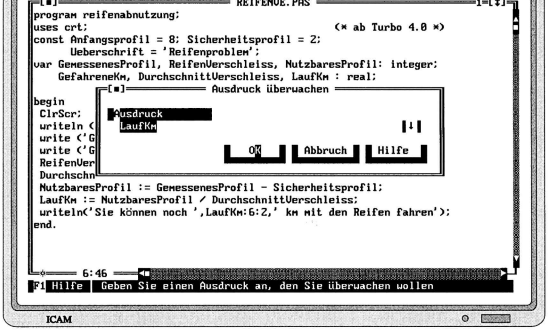
> Der Tracebalken wird entsprechend der im Programm festgelegten Ablauflogik gesteuert.

Variablen inspizieren

Bei der Ausführung des «verbesserten» Programms wird Ihnen am Bildschirm ein falsches Ergebnis angezeigt.

```
Reifenproblem
Gefahrene Kilometer? 25000
Gemessenes Profil in mm? 3
Sie können noch    0.00 km mit den Reifen fahren
```

Um die Anweisung herauszufinden, in der der falsche Wert gebildet wird, lassen Sie sich die Inhalte der betreffenden Variablen anzeigen. Diese Inspektion von Variablen geschieht im Beobachtungsfenster. Um eine Variable unter Beobachtung zu stellen, positionieren Sie den Cursor auf den Namen der betreffenden Variablen und betätigen die Tastenkombination Strg F7. Das können Sie vor Beginn des Tracens, aber auch während des Testlaufs machen.



Auf Strg F7 reagiert das System mit einem *Ausdruck-überwachen*-Fenster, in dessen Eingabefeld der durch die Cursorposition gekenn-

zeichnete Variablenname übernommen ist. Durch ↵ oder durch Betätigen des Aktionsschalters *OK* wird die Variable in das Überwachungsfenster übernommen. Wenn Sie nun weitere Variablen überwachen wollen, stehen Ihnen zwei Möglichkeiten offen.

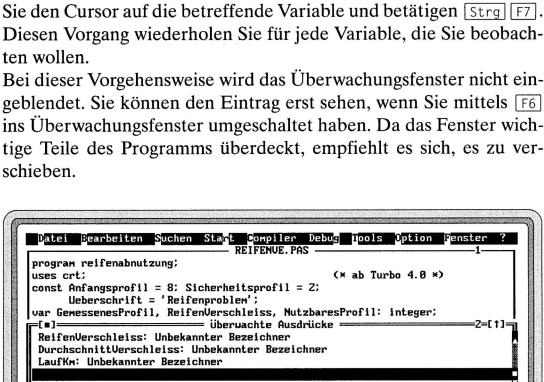
1. Möglichkeit

Sie betätigen die Taste Einfg und geben in die Eingabezeile unter *Ausdruck* den Namen der betreffenden Variablen ein. Sie können sich auch der Eingabeaufzeichnungsliste bedienen, wenn Sie bereits während der Arbeitssitzung Variablennamen eingegeben hatten.

2. Möglichkeit

Sie schalten mittels ⇧ F6 zurück in das Editfenster, positionieren Sie den Cursor auf die betreffende Variable und betätigen Strg F7. Diesen Vorgang wiederholen Sie für jede Variable, die Sie beobachten wollen.

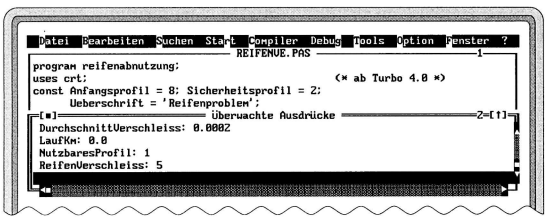
Bei dieser Vorgehensweise wird das Überwachungsfenster nicht eingeblendet. Sie können den Eintrag erst sehen, wenn Sie mittels F6 ins Überwachungsfenster umgeschaltet haben. Da das Fenster wichtige Teile des Programms überdeckt, empfiehlt es sich, es zu verschieben.



Die Meldung «unbekannter Bezeichner» hinter den Variablennamen im Überwachungsfenster besagt, daß diese Variablen dem System zum gegenwärtigen Zeitpunkt, hier vor Beginn des Programmlaufs, noch nicht bekannt sind. Sobald das Programm mittels F7 gestartet wird, wird der Inhalt der Speicherplätze angezeigt.

Bei der schrittweisen Ausführung des Programms erkennen Sie, daß

die Variable LaufKm stets den Wert 0.0 aufweist. Wenn Sie die Variableninhalte unmittelbar nach der Wertzuweisung auf LaufKm im Beobachtungsfenster betrachten, sehen Sie, daß der Inhalt der Variablen NutzbaresProfil stets 0 ist. Ihr Inhalt wird nämlich erst im nächsten Befehl berechnet. Folglich kann LaufKm auch nur den Wert 0 erhalten. Der Fehler ist somit gefunden.



Aktivieren Sie das Editfenster, und beheben Sie den Fehler. Wenn Sie den Programmlauf erneut starten, fragt Sie das System, ob eine erneute Übersetzung vorgenommen werden soll. Da es sich in diesem Fall um einen Logikfehler handelt, bejahen Sie die Frage.

Zur Verwaltung des Überwachungsfensters stehen Ihnen neben Einfg und Strg F7 noch weitere Tasten zur Verfügung:

Entf — Löschen der Variablen, auf der der Cursorbalken steht, im Überwachungsfenster.

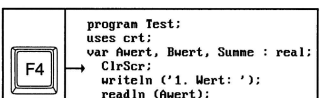
↵ — Bearbeiten des Ausdrucks, auf dem der Cursorbalken steht.

Trace-Lauf abbrechen

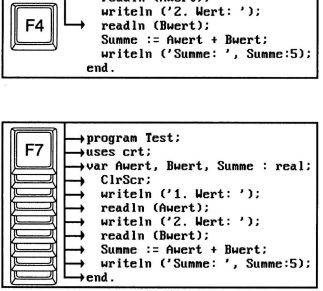
Haben Sie den Fehler gefunden, so macht es möglicherweise keinen Sinn, das Programm noch weiter im Einzelschritt bis zum Abschluß auszuführen. Mittels Strg F2 brechen Sie den Trace-Lauf ab. Wollen Sie das Programm ohne Trace bis zum Ende durchlaufen lassen, geben Sie Strg F9 ein.

Trace ab einer bestimmten Anweisung

Sie brauchen beim Überprüfen eines Programms nicht sämtliche Anweisungen, die Sie bereits als richtig erkannt haben, schrittweise zu durchlaufen. Vielmehr können Sie dem System mitteilen, daß Sie erst ab einer bestimmten Anweisung tracen möchten. Dazu stellen Sie den Cursor auf die Anweisung, ab der mit der schrittweisen Ausführung begonnen werden soll, und starten das Programm mit F4. Das Programm läuft bis zu dieser Programmzeile im «Normalbetrieb». Diese Trace-Art wird «streckenweise» Ausführung genannt.



streckenweise Ausführung



schrittweise Ausführung

Das Programm wird bis zu einem temporären Haltepunkt ohne Unterbrechung ausgeführt. An dem Haltepunkt können Sie durch einen Blick in die Inhalte der Variablen die erreichten Zwischenergebnisse kontrollieren und anschließend mittels F7 das Programm im Einzelschrittverfahren weiterlaufen lassen. Sie können aber auch die «streckenweise» Ausführung fortsetzen. Dazu positionieren Sie den Cursor auf eine der folgenden Anweisungen, bis zu der der Programmlauf mittels F4 fortgeführt werden soll. Der Cursor muß jedoch auf einen ausführbaren Befehl gestellt werden.

Bei umfangreichen Programmen ist die abschnittsweise Ausführung vielleicht doch zu umständlich. Deshalb gibt es im Turbo-Pascal-

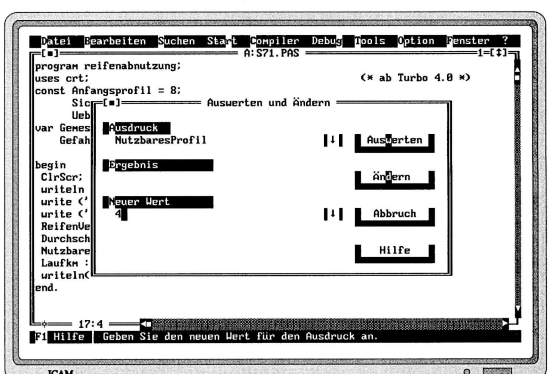
System die Möglichkeit, mittels Strg F8 dauerhafte Haltepunkte zu setzen, bis zu denen das durch Strg F9 gestartete oder fortgesetzte Programm jeweils läuft.

3.6.4 Variableninhalte verändern

Der Programmtest läßt sich verkürzen, wenn man geeignete Testbedingungen rasch und einfach erzeugen kann. Dazu bietet das Pascal-System die Funktion *Auswerten und Ändern* an. Diese Funktion gestattet es Ihnen, Variablen während eines Trace-Laufes «von außen» neue Werte zuzuweisen. Die Vorgehensweise sei am obigen Programmbeispiel aufgezeigt. Vor Ausführung der Anweisung

```
LaufKm: = NutzbaresProfil / DurchschnittVerzeichnis;
```

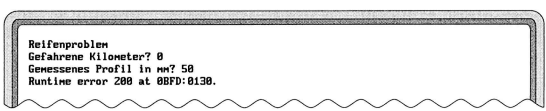
stellen Sie den Cursor auf die Variable NutzbaresProfil und betätigen die Tastenkombination Strg F4. Auf dem Bildschirm wird das Fenster *Auswerten und Ändern* angezeigt, in dessen Eingabezeile bereits der Variablenname NutzbaresProfil übernommen wurde.



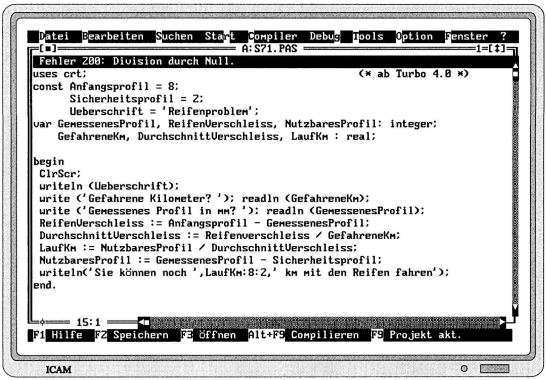
Schalten Sie nun beispielsweise durch Alt N weiter auf das Eingabefeld *Neuer Wert*, und geben Sie dort den Wert 4 ein. Abschluß der Eingabe durch ↵. In der Zeile *Ergebnis* wird der neue Inhalt der Variablen NutzbaresProfil angezeigt. Mit Esc verlassen Sie die Auswerten-Funktion und kehren in das Programm zurück.

3.6.5 Laufzeitfehler

Ein Laufzeitfehler kommt zwar nicht so häufig vor wie die beiden anderen Fehlerarten, ist aber dennoch nicht minder ärgerlich. Damit Sie sehen, was ein Laufzeitfehler ist, starten Sie das Programm Reifenabnutzung und geben den Wert 0 in die Variable GefahreneKm ein. Für GemessenesProfil können Sie einen beliebigen Wert eintasten. Es kommt aber zu keiner weiteren Berechnung, weil das Programm «abstürzt». Am Bildschirm erscheint die Fehlermeldung



Wenn Sie in den Editbildschirm umschalten, wird der Fehler erklärt.



In der Anweisung

```
DurchschnittVerschleiss := ReifenVerschleiss / GefahreneKm;
```

sollte der Rechner durch null dividieren. Eine Division durch null ist nicht erlaubt. Das widerspricht den Regeln der Mathematik. Da das System diese Rechenoperation nicht ausführen kann, bricht es den Programmlauf ab.
Ein Laufzeitfehler liegt also immer dann vor, wenn die Ausführung eines logisch korrekten Programms abgebrochen wird, weil eine Anweisung nicht ausgeführt werden kann. Der Programmierer muß, wie später noch gezeigt wird, durch eine geeignete Programmgestaltung dafür sorgen, daß möglichst keine Laufzeitfehler auftreten.

3.7 Dokumentation

Ist das Programm syntaktisch und logisch in Ordnung, muß abschließend beschrieben werden, was es leistet und wie es bedient wird. Diese Beschreibung nennt man Dokumentation. Zur Dokumentation gehört ferner das Programmlisting.

Die Bedieneranweisung ist wichtig für denjenigen, der das Programm nutzt, ohne den genauen Ablauf des Programms zu kennen. Falls das Programm später einmal geändert werden soll, bildet die Dokumentation eine wichtige Arbeitsunterlage für den Programmierer. Er kann aus den Unterlagen erkennen, wie und an welcher Stelle das Programm zu ändern ist.

Für Unternehmen ist die Dokumentation auch noch aus einem anderen Grunde von Bedeutung. Werden mit dem Programm handels- und steuerrechtlich wichtige Daten verarbeitet, so kann an Hand der Dokumentation nachgeprüft werden, ob die Datenverarbeitung nach handels- und steuerrechtlichen Gesichtspunkten korrekt vorgenommen wurde.

In dem Reifenbeispiel kann die Dokumentation kurz gehalten werden. Sie berücksichtigt die Arbeitsumgebung der Version 3.

Leistung des Programms

Mit diesem Programm wird berechnet, über wie viele Kilometer die Reifen eines Autos noch genutzt werden können. Es wird davon ausgegangen, daß die anfängliche Profiltiefe 8 mm beträgt. Als Mindestprofiltiefe werden 2 mm angenommen.

Bedienungsanleitung

- Voraussetzungen
  (a) Der Computer ist betriebsbereit. Das Betriebssystem MS-DOS ist geladen.
  (b) Legen Sie die Arbeitsdiskette in das Laufwerk A:.
  (c) Rufen Sie mittels der Eingabe turbo ↵ das Turbo-Pascal-System auf.

  Beantworten Sie die Systemfrage «Include error messages» mit Y.
- Start des Programms

  Nach der Anzeige des Hauptmenüs starten Sie das Programm mit R. Das System meldet sich mit «Work-file name:».

  Geben Sie ein: A:reifen.PAS ↵
- Programmdurchführung
  (a) Eingabe der gefahrenen Kilometer, zum Beispiel 25000. Abschluß der Eingabe mit ↵.
  (b) Eingabe der gemessenen Profiltiefe in mm, zum Beispiel 3. Abschluß der Eingabe mit ↵.

(c) Am Bildschirm wird das Ergebnis angezeigt, zum Beispiel: Sie können noch 5000.00 km mit den Reifen fahren.

(d) Beenden der Arbeit: Betätigen Sie die Taste [Q] (für quit). Das Betriebssystem meldet sich mit C:>.

Entnehmen Sie die Diskette dem Laufwerk.

3.8 Zusammenfassung

- Programme sind systematisch zu entwickeln!
- Die Schritte zur Erstellung eines Programms sind:
  (1) Klärung der Problemstellung
  (2) Analyse der Daten
  Ausgabedaten
  Eingabedaten
  Beziehungen zwischen Ein- und Ausgabedaten
  (3) Entwicklung des Algorithmus
  (4) Formulierung des Programms
  (5) Eingabe und Test des Programms
  Syntaxtest
  Logiktest
  (7) Dokumentation
- Der integrierte Debugger gestattet die schrittweise Ausführung eines Programms, die Inspektion und die Manipulation von Variableninhalten.
- Unter Trace versteht man die schrittweise Ausführung eines Programms.
- Trace läuft auf der Ebene des Quellprogramms. Man kann die Ausführung einer Anweisung unmittelbar nachvollziehen.
- Die schrittweise Programmausführung wird durch [F7] gestartet.
- Mittels [F4] wird ein temporärer Haltepunkt gesetzt, bis zu dem ein Programm «ohne Zwischenhalt» durchgeführt wird.
- Durch [Strg] [F8] werden dauerhafte Haltepunkte gesetzt.
- Im Überwachungsfenster werden die Inhalte von Variablen angezeigt, die mittels [Strg] [F7] in das Fenster eingestellt worden sind.

3.9 Übungen

1. Aufgabe
In einem Programm müssen Zahlen eingegeben werden, die zwischen 10000 und 40000 liegen. Von welchem Datentyp muß die Eingabevariable sein (x)?

( ) integer
( ) real
( ) char

2. Aufgabe
Ergänzen Sie die fehlenden Datentypangaben und Druckmasken in dem nachstehenden Programm. Die Ausgabedaten sollen mit 6 Stellen ausgedruckt werden. Wenn es sich um Real-Zahlen handelt, so sollen in den 6 Stellen auch die zwei Nachkommastellen enthalten sein.

```
program test;
var a,b: integer; c: real;
    d:.....;
    e:.....;
begin
    readln (a); readln (b); readln (c);
    d := a / b;
    e := a * c;
    writeln (d:..., e:...);
end.
```

3. Aufgabe
Auf manchen Rechnungen finden Sie den Satz: «Im Rechnungsbetrag sind 15% Mehrwertsteuer enthalten.»
Entwickeln Sie ein Programm, mit dem der Mehrwertsteuerbetrag aus dem Rechnungsbetrag berechnet wird!

Eingabedaten:	Rechnungsbetrag
Ausgabedaten:	Mehrwertsteuerbetrag
	Nettobetrag

Lösungshinweis
Die Mehrwertsteuer wird mit 15% auf den Nettobetrag aufgeschlagen. Der Rechnungsbetrag, der sich aus Nettobetrag + Mehrwertsteuerbetrag errechnet, entspricht also 115% vom Nettobetrag.

Beispiel:

Nettobetrag	50,00 DM	100 %
+ 15 % Mehrwertsteuer	7,50 DM	15 % vom Nettobetrag
Rechnungsbetrag	57,50 DM	115 %

4. Aufgabe
Der Debugger hilft, Fehler zu finden. Um welche Art von Fehlern handelt es sich?

5. Aufgabe
Wie wird ein Trace-Lauf gestartet?

6. Aufgabe
Welcher Unterschied besteht in der Funktionsweise von F4 und F7?

7. Aufgabe
Was leistet das Überwachungsfenster?

8. Aufgabe
Wie wird eine Variable unter Beobachtung gestellt?

3.10. Lösungen

1. Aufgabe
Die Eingabevariable muß vom Typ Real sein, weil in einer Integer-Variablen nur Zahlen von –32768 bis 32767 gespeichert werden können.

2. Aufgabe

```
program test;
var a, b: integer; c: real;
    d: real;
    e: real;
begin
    readln (a); readln (b); readln (c);
    d := a / b;
    e := a * c;
    writeln (d:10:2, e:10:2);
end.
```

3. Aufgabe

Problemanalyse

Der Rechnungsbetrag beträgt 115% des Nettobetrags. Um aus dem Rechnungsbetrag den Nettobetrag zu errechnen, ist der Rechnungsbetrag mit 100 zu multiplizieren und anschließend durch 115 zu dividieren.

Datenanalyse

		Datentyp
Zusammenstellung der Ausgabedaten:	Nettobetrag	real
	MWST_Betrag	real

Zusammenstellung der Eingabedaten: Rechnungsbetrag real

Beziehungen zwischen Ein- und Ausgabedaten:

Nettobetrag := Rechnungsbetrag * 100 / 115
MWST_Betrag:= Rechnungsbetrag – Nettobetrag

Wird der Mehrwertsteuersatz unter dem Namen Steuersatz = 15 als Konstante vereinbart, lautet die Berechnung des Nettobetrags:

Nettobetrag := Rechnungsbetrag * 100 / (100 + Steuersatz)

Algorithmus

Algorithmus Nettobetragsberechnung
Konstante Steuersatz = 15
Eingabe Rechnungsbetrag
Berechne Nettobetrag
Berechne MWST_Betrag
Ausgabe Nettobetrag
Ausgabe MWST_Betrag
Ende

Programm

```
program Nettobetragsberechnung;
uses crt;                                          (* ab Version 4 *)
const Steuersatz = 15;
var Rechnungsbetrag, MWST_Betrag, Nettobetrag: real;
begin
  ClrScr;
  write ('Eingabe Rechnungsbetrag: ');
  readln (Rechnungsbetrag);
  Nettobetrag := Rechnungsbetrag * 100/ (100 + Steuersatz);
  MWST_Betrag := Rechnungsbetrag - Nettobetrag;
  writeln;
  writeln ('Nettobetrag:',Nettobetrag:8:2);
  writeln ('MWST-Betrag:',MWST_Betrag:8:2);
end.
```

4. Aufgabe

Syntaktische Fehler entdeckt der Compiler. Logische Fehler kann nur der Programmierer entdecken. Bei der Suche nach logischen Fehlern kann ein Debugger hilfreich sein.

5. Aufgabe

Mittels F7 wird die schrittweise Programmausführung gestartet.

6. Aufgabe

F7 veranlaßt eine schrittweise Ausführung des Programms. Das Programm bleibt nach jeder Anweisung stehen und wartet auf den nächsten Impuls. Bei F4 wird das Programm ohne Unterbrechung bis zu einem vorher gesetzten Haltepunkt ausgeführt.

7. Aufgabe

Im Überwachungsfenster werden Inhalte von Variablen angezeigt.

8. Aufgabe

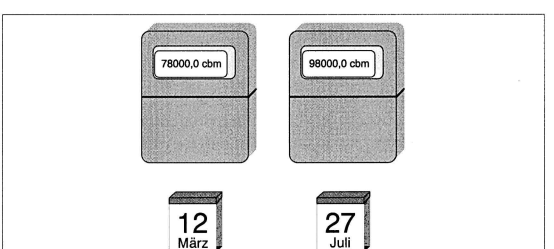
Man positioniert den Cursor auf die betreffende Variable und betätigt die Tastenkombination Strg F7.

4 AUSWAHLANWEISUNGEN

Die bislang behandelten Programme sind dadurch gekennzeichnet, daß die Anweisungen linear von oben nach unten ausgeführt werden. Die oberste Anweisung wird als erste, die unterste Anweisung wird als letzte bearbeitet. Dabei wird jede Anweisung genau einmal ausgeführt.
In diesem Kapitel werden Problemstellungen behandelt, bei denen Anweisungen in Abhängigkeit von einer Bedingung ausgeführt bzw. auch nicht ausgeführt werden.

4.1 Einseitige Auswahl: IF – THEN

Programmbeispiel **Gaspreis**
Gaszähler sind mit fünfstelligen Zählwerken ausgerüstet. Damit können sie Verbrauchswerte zwischen 00000 und 99999 Kubikmeter (cbm) anzeigen. Wenn die Verbrauchswerte 99999 cbm übersteigen, beginnen die Zähler wieder von vorn zu zählen.



Um den Preis für das verbrauchte Gas zu ermitteln, wird zunächst der Gasverbrauch errechnet. Dazu wird vom aktuellen Zählerstand der Zählerstand subtrahiert, der beim vorherigen Ablesetag festgestellt wurde. Der Gaspreis ergibt sich dann aus der Multiplikation des Gasverbrauchs mit dem cbm-Preis.

	Ablesetag 02.01.92	Ablesetag 25.11.92	Verbrauch (cbm)	cbm-Preis	Gas-Preis
Zähler-stand	78000	98000	20000	0.23	4600.00

Die Gasabrechnung wird heute in der Regel mit einem Computer erstellt. Dabei sei folgendes Programm im Einsatz. Untersuchen Sie, ob das Programm für die Berechnung des Gaspreises geeignet ist. Der Preis pro Kubikmeter (cbm) Gas soll 0.23 DM betragen.

```
program Gaspreis;
uses crt;                                          (* ab Version 4 *)
const cbmPreis = 0.23;
var AltZaehler, NeuZaehler, Verbrauch, Preis: real;

begin
  ClrScr;
  write ('Eingabe alter Zählerstand: '); readln (AltZaehler);
  write ('Eingabe neuer Zählerstand: '); readln (NeuZaehler);
  Verbrauch := NeuZaehler - AltZaehler;
  Preis := Verbrauch * cbmPreis;
  writeln ('Der Gaspreis beträgt ',Preis:8:2);
end.
```

Wenn Sie das Programm eingegeben haben, können Sie die Wirkungsweise des Programms leicht an Hand der untengenannten Zahlen überprüfen.

Wertbelegungstabelle

Falls Sie keinen Rechner zur Verfügung haben, können Sie die Funktionsweise des Programms mit Hilfe einer Wertbelegungstabelle testen.

Eingabewerte:	Alter Zählerstand	= 78000
	Neuer Zählerstand	= 98000

Anweisungen	Variablen			
Konstante: cbmPreis = 0.23	Alt-Zaehler	Neu-Zaehler	Ver-brauch	Preis
readln (AltZaehler)	78000			
readln (NeuZaehler)		98000		
Verbrauch := NeuZaehler – AltZaehler			20000	
Preis := Verbrauch * cbmPreis				4600

Bei einem erneuten Programmlauf sollen folgende Werte eingegeben werden:

Eingabewerte: Alter Zählerstand = 98000
Neuer Zählerstand = 18000

Anweisungen	Variablen			
Konstante: cbmPreis = 0.23	Alt-Zaehler	Neu-Zaehler	Ver-brauch	Preis
readln (AltZaehler)	98000			
readln (NeuZaehler)		18000		
Verbrauch := NeuZaehler – AltZaehler			–80000	
Preis := Verbrauch * cbmPreis				–18400

Im ersten Durchlauf hat der Computer richtig gerechnet und ein vernünftiges Ergebnis ausgegeben. Im zweiten Durchlauf hat der Computer auch richtig gerechnet. Das Ergebnis ist aber nicht sinnvoll. Es gibt nämlich keinen negativen Gasverbrauch und keinen negativen Preis.
Daß es im zweiten Durchlauf zu unsinnigen Ergebnissen kommt, liegt am Programm. Es berücksichtigt nicht, daß der Gaszähler «überspringen» kann und wieder bei 00000 zu zählen beginnt. Der «echte» neue Zählerstand beläuft sich eigentlich auf 118000.
Damit die Verbrauchsabrechnung richtig läuft, muß der neue Zählerstand um 100000 erhöht werden, wenn der neue Zählerstand kleiner ist als der alte Zählerstand. Diese bedingte Zählerstandserhöhung wird wie folgt ausgedrückt:

WENN NeuZaehler < AltZaehler DANN NeuZaehler := NeuZaehler + 100000

IF-THEN-Anweisung

In Pascal lautet diese bedingte Zählerstandserhöhung:

```
if NeuZahler < AltZaehler then NeuZaehler := NeuZaehler +
100000.0
```

Die WENN-DANN-Anweisung wird in Pascal mit dem Schlüsselwort **if** eingeleitet. Dahinter folgt eine Bedingung oder ein Bedingungsausdruck. Daran schließt sich das Schlüsselwort **then** an. Hinter then wird die Anweisung angegeben, die auszuführen ist, wenn die Bedingung oder der Bedingungsausdruck erfüllt ist.
Ob der Wert in der Variablen NeuZaehler um 100000.0 erhöht wird, hängt also ab von der Bedingung NeuZaehler < AltZaehler, die hinter dem if steht. Die Wertzuweisung

```
NeuZaehler := NeuZaehler + 100000.0
```

hinter then wird vorgenommen, wenn die Bedingung erfüllt ist. Wenn die Bedingung nicht erfüllt ist, unterbleibt die Zählerstandserhöhung.

Anmerkung: Da die konstante Zahl 100000 im Rechner nicht als ganze Zahl dargestellt werden kann, muß sie durch 100000.0 als Real-Zahl gekennzeichnet werden.

Diese Auswahlanweisung IF – THEN ist nun in das obige Programm Gaspreis einzubeziehen. Die IF-Anweisung muß vor der Anweisung stehen, in der der Verbrauch berechnet wird. Das Programm lautet somit:

```
program Gaspreis;
uses crt;                                    (* ab Version 4 *)
const cbmPreis = 0.23;
var AltZaehler, NeuZaehler, Verbrauch, Preis: real;

begin
  ClrScr;
  write ('Eingabe alter Zählerstand: '); readln (AltZaehler);
  write ('Eingabe neuer Zählerstand: '); readln (NeuZaehler);
  if NeuZaehler < AltZaehler
    then NeuZaehler := NeuZaehler + 100000.0;
  Verbrauch := NeuZaehler - AltZaehler;
  Preis := Verbrauch * cbmPreis;
  writeln ('Der Gaspreis beträgt ',Preis:8:2);
end.
```

Wenn Sie die erste Version des Programms Gaspreis in den Computer eingegeben haben, sollten Sie es jetzt auch um die IF-Anweisung erweitern. Sie können sich durch weitere Programmläufe die Wirkungsweise des Programms verdeutlichen.
Auch an Hand der Wertbelegungstabellen können Sie die Wirkungsweise der IF-Anweisung nachvollziehen.

Eingabewerte: Alter Zählerstand = 78000
Neuer Zählerstand = 98000

Anweisungen	**Variablen**			
Konstante: cbmPreis = 0.23	Alt- Zaehler	Neu- Zaehler	Ver- brauch	Preis
readln (AltZaehler)	78000			
readln (NeuZaehler)		98000		
if NeuZaehler < AltZaehler **then** NeuZaehler := NeuZaehler + 100000.0				
Verbrauch := NeuZaehler – AltZaehler			20000	
Preis := Verbrauch * cbmPreis				4600

Da die Bedingung NeuZaehler < AltZaehler nicht erfüllt ist, unterbleibt die Erhöhung des Wertes in NeuZaehler. Das Programm wird also unmittelbar mit der Verbrauchsberechnung fortgesetzt.

Erneuter Programmlauf:

Eingabewerte: Alter Zählerstand = 98000
Neuer Zählerstand = 18000

Anweisungen	Variablen			
Konstante: cbmPreis = 0.23	Alt-Zaehler	Neu-Zaehler	Ver-brauch	Preis
readln (AltZaehler)	98000			
readln (NeuZaehler)		18000		
if NeuZaehler < AltZaehler **then** NeuZaehler := NeuZaehler + 100000.0		118000		
Verbrauch := NeuZaehler – AltZaehler			520000	
Preis := Verbrauch * cbmPreis				4600

Da die Bedingung NeuZaehler < AltZaehler erfüllt ist, wird der Inhalt von NeuZaehler um 100000.0 erhöht. Nun erfolgt also die Berechnung des Verbrauchs und des Preises korrekt.

Trace

Der Programmlauf können Sie ab der Turbo-Pascal-Version 5.x auch am Rechner nachvollziehen. Dazu muß das Programm im Editfenster stehen. Damit Sie die Variableninhalte während des Programmlaufs verfolgen können, stellen Sie zunächst die Variable AltZaehler mittels Strg F7 in das Überwachungsfenster. In das eingeblendete Überwachungsfenster tragen Sie die Variablenbezeichnungen NeuZaehler und Verbrauch ein. Dazu öffnen Sie mittels Einfg das Eingabefenster, schreiben NeuZaehler in die Eingabezeile und schließen die Eingabe mittels ↵ ab. Auf die gleiche Weise tragen Sie die Variablenbezeichnung Verbrauch in das Überwachungsfenster ein. Damit das Fenster nicht den Programmtext verdeckt, verschieben Sie es nach rechts oben. Nun veranlassen Sie mittels F7, daß der Rechner nach der Ausführung einer Anweisung anhält. Betätigen Sie erneut F7, wird die jeweils logisch nächste Anweisung ausgeführt. Bei dieser schrittweisen Ausführung können Sie nun sehr genau erkennen, welche Anweisungen ausgeführt bzw. nicht ausgeführt werden.

Syntax
Die allgemeine Form der IF-THEN-Anweisung lautet

if Bedingungsausdruck **then** Anweisung;

Die IF-Anweisung gilt als eine Anweisung. Deshalb darf innerhalb der IF-Anweisung kein Semikolon stehen.

Vor then darf kein Semikolon stehen!

Bedingungsausdruck
Über einen Bedingungsausdruck wird gesteuert, in welcher Reihenfolge Anweisungen ausgeführt werden. In dem obigen Programmbeispiel Gaspreis ist die Verarbeitung abhängig von dem Bedingungsausdruck

NeuZaehler < AltZaehler

Der Bedingungsausdruck besteht hier aus dem Vergleich zweier Variablen. Der Inhalt der Variablen, die links vom Vergleichsoperator «<» steht, wird mit dem Inhalt der Variablen verglichen, die rechts vom Vergleichsoperator steht. Als Vergleichsoperatoren kennt Pascal:

Vergleichsoperator	Bedeutung
>	größer
<	kleiner
=	gleich
>=	größer gleich
<=	kleiner gleich
<>	ungleich

Beispiele	Bedeutung
AWert < 0	Inhalt von AWert kleiner 0
2*B <> 3*C	Zweifaches von B ungleich Dreifaches von C
sqr(p/2) – q > 0	$(p/2)^2 - q$ größer 0
	(* sqr = square = Quadrat von *)

In einen Vergleich können also auch arithmetische Operationen einbezogen werden.

Ein Vergleich hat immer genau ein Ergebnis. Er kann das Ergebnis **true** (deutsch: wahr) oder das Ergebnis **false** (deutsch: falsch) haben. Da ein Bedingungsausdruck stets das Ergebnis true oder false (wahr oder falsch) hat, kann eine Variable, die vom Datentyp Boolean ist, unmittelbar als Bedingungsausdruck verwendet werden. Sie weist nämlich ebenfalls stets nur einen der beiden Werte true oder false auf.

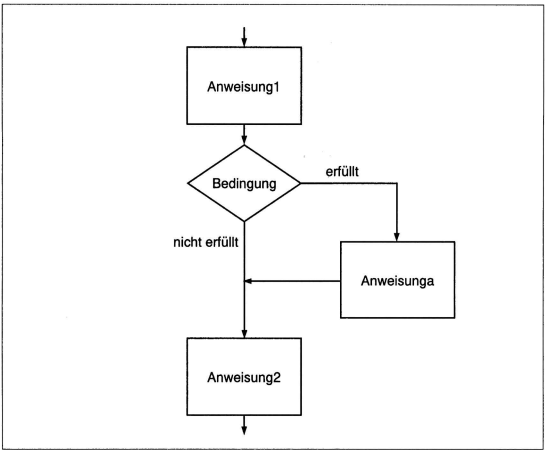
Beispiel:

Abfrage auf true:	**Abfrage auf false:**
...	...
var Richtig: boolean;	var Richtig: boolean;
begin	begin
...	...
if Richtig then Anweisung;	if Richtig = false then Anweisung;
...	...
end.	end.

Ablaufsteuerung

An dem Programmbeispiel Gaspreis konnten Sie erkennen, wie der Programmablauf bei einer IF-THEN-Anweisung gesteuert wird. Ist der Bedingungsausdruck hinter if erfüllt, wird die Anweisung hinter then ausgeführt. Ist der Bedingungsausdruck nicht erfüllt, übergeht das Programm die Anweisung hinter then und fährt mit der Anweisung fort, die der IF-Anweisung folgt. Das verdeutlicht auch die graphische Darstellung der einfachen Auswahl in Form des Programmablaufplans und des Struktogramms.

Darstellung der einfachen Auswahl in Form eines Programmablaufplans:



Darstellung der einfachen Auswahl in Form eines Struktogramms:

Anweisung1	
Bedingung erfüllt	nicht erfüllt
Anweisunga	. / .
Anweisung2	

Verbundanweisung

Die strenge Syntaxregel, daß innerhalb einer IF-Anweisung kein Semikolon stehen darf, führt zu einer Komplikation. Was ist zu tun, wenn auf dem THEN-Weg mehrere Anweisungen zu bearbeiten sind? Beispielsweise soll immer dann, wenn der Gaszähler wieder von vorn zu zählen begonnen hat, auf dem Bildschirm der Hinweis «Zählerübersprung» erscheinen.

```
if NeuZahler < AltZaehler
  then
    NeuZaehler := NeuZaehler + 100000.0;
    writeln ('Zählerübersprung');
Verbrauch := NeuZaehler – AltZaehler;
```

Der THEN-Weg besteht nun aus zwei Anweisungen. Wenn Sie aber das Programm in der obigen Fassung in den Rechner eingeben, erscheint bei jedem Programmdurchlauf die Meldung «Zählerübersprung» auf dem Bildschirm. Der Computer nimmt auf Grund der Syntaxregel an, daß die IF-Anweisung durch das Semikolon hinter

```
NeuZaehler := NeuZaehler + 100000.0;
```

beendet ist.
Um das zu vermeiden, muß man die beiden Anweisungen, die zum THEN-Weg gehören, mit Hilfe von begin und end einklammern. Sie werden dadurch zu einer Verbundanweisung zusammengefaßt. Der Programmausschnitt lautet dann:

```
if NeuZaehler < AltZaehler then
  begin
    NeuZaehler := NeuZaehler + 100000.0;  } Verbund-
    writeln ('Zählerübersprung');         } anweisung
  end;
Verbrauch := NeuZaehler – AltZaehler;
```

Nun erstreckt sich die IF-Anweisung bis zum Semikolon hinter dem end. Das Semikolon hinter writeln ('Zählerübersprung') kann stehenbleiben, obwohl es überflüssig ist. Das System interpretiert es als Leeranweisung.
Unter Einbeziehung der Verbundanweisung lautet die IF-THEN-Anweisung allgemein:

```
if  Bedingungsausdruck
    then begin
           Anweisung;
           Anweisung;
           ...
    end;
```

Sind auf dem THEN-Weg mehrere Anweisungen zu bearbeiten, so sind diese Anweisungen durch begin... end zu einer Verbundanweisung zusammenzufassen!

4.1.1 Zusammenfassung

- Mit Hilfe einer Wertbelegungstabelle kann man Programme im «Trockentest» überprüfen.
- Die einfache Auswahl lautet:

  if Bedingungsausdruck then AnweisungA;

  AnweisungA wird nur dann bearbeitet, wenn der Bedingungsausdruck wahr ist.
- Folgen auf dem THEN-Weg mehrere Anweisungen, so sind sie durch begin und end zu einer Verbundanweisung zusammenzufassen.
- Ein Bedingungsausdruck ist stets wahr oder falsch.
- Ein Bedingungsausdruck besteht aus dem Vergleich von zwei Werten oder aus der Abfrage des Wahrheitswerts einer Boolean-Variable.

4.1.2 Übungen

1. Aufgabe

Mit einem Programm wird die Wurzel aus einer eingegebenen Zahl errechnet. Die Wurzelberechnung darf nur dann vorgenommen werden, wenn die Zahl größer null ist.

Hinweis: Die Wurzel einer Zahl wird mit Hilfe der Standardfunktion sqrt (englisch: squareroot = Quadratwurzel) berechnet.

sqrt(9) = 3

Ergänzen Sie das Programm Wurzelberechnung um die Auswahlanweisung.

```
program Wurzelberechnung;
var Zahl,Wurzel: real;
begin
    write ('Eingabe einer Zahl: '); readln (Zahl);
    if____
            ____
            Wurzel := sqrt(Zahl);
            writeln (Wurzel:10:5);
            ____
    end.
```

2. Aufgabe

Nach der deutschen Zinsrechnung besteht der Zinsmonat aus 30 Tagen. Wenn Sie beispielsweise am 31.01. einen Betrag auf Ihr Sparkonto einzahlen, werden die Zinsen ab dem 30.01. berechnet für den Zeitraum bis zum Ende des Jahres (30.12.).
Entwickeln Sie ein Programm, mit dem die Zinsen für einen Betrag, der auf ein Sparkonto eingezahlt wird, vom Tag der Einzahlung bis zum Jahresende berechnet werden. Der Zinssatz sei mit 3% konstant.

Problemanalyse

Die Formel zur Berechnung der Tageszinsen lautet:

$$\text{Zinsen} = \frac{\text{Betrag} * \text{Zinstage} * \text{Zinssatz}}{36000}$$

Datenanalyse

		Datentyp
Festlegung der Ausgabedaten:	Zinsen	.....
Festlegung der Eingabedaten:	Betrag	.....
	Einzahlungs-datum	.....

Bei der Festlegung des Datentyps für das Einzahlungsdatum ergeben sich Schwierigkeiten. Die übliche Schreibweise für ein Datum, beispielsweise 31. 01., kann nicht angewendet werden, weil es in dieser Form nicht in einer numerischen Variablen abgebildet wer-

den kann. Deshalb wird das Einzahlungsdatum aufgeteilt in zwei Eingaben:

Tag .....
Monat .....

Beziehungen zwischen Ein- und Ausgabedaten:

ZinsTage := (12 – .....)*30 + (30 – .....)

Anmerkung: Die Monatedifferenz wird in eine Tagedifferenz umgerechnet, indem sie mit 30 multipliziert wird.

Zinsen := ..... / 36000.0

Algorithmus

Algorithmus TageszinsBerechnung	
Konstante Zinssatz = 3	
Eingabe Betrag	
Eingabe Tag	
Eingabe Monat	
Wenn Tag ____ ja	nein
Tag := 30	./.
Berechne ZinsTage	
Berechne Zinsen	
Ausgabe Zinsen	
Ende	

4.1.3 Lösungen

1. Aufgabe

```
program Wurzelberechnung;
var Zahl, Wurzel: real;
begin
  write ('Eingabe einer Zahl: ');
  readln (Zahl);
  if Zahl > 0
     then begin
            Wurzel := sqrt(Zahl);
            writeln (Wurzel:10:5);
          end;
end.
```

2. Aufgabe

Datenanalyse

		Datentyp
Festlegung der Ausgabedaten:	Zinsen	**real**
Festlegung der Eingabedaten:	Betrag	**real**
	Tag	**integer**
	Monat	**integer**

Beziehungen zwischen Ein-/Ausgabedaten:

ZinsTage := (12 – Monat)*30 + (30 – Tag)

Zinsen := **Betrag** * **Tage** * **Zinssatz** / 36000.0

Algorithmus

Algorithmus TageszinsBerechnung	
Konstante Zinssatz = 3	
Eingabe Betrag	
Eingabe Tag	
Eingabe Monat	
Wenn Tag = 31	
Ja	nein
Tag := 30	./.
Berechne ZinsTage	
Berechne Zinsen	
Ausgabe Zinsen	
Ende	

Programm

```
program TageszinsBerechnung;
uses crt;                                          (* ab Version 4 *)
const Zinssatz = 3;
var Tag, Monat, ZinsTage: integer;
    Betrag, Zinsen: real;
begin
  ClrScr;
  write ('Eingabe Betrag: '); readln (Betrag);
  write ('Eingabe Tag   : '); readln (Tag);
  write ('Eingabe Monat : '); readln (Monat);
  if Tag = 31 then Tag := 30;
  ZinsTage := (12 - Monat)*30 + 30 - Tag;
  Zinsen := Betrag * Zinssatz * Tage / 36000.0;
  writeln ('Die Zinsen betragen:', Zinsen:8:2);
end.
```

4.2 Zweiseitige Auswahl: IF – THEN – ELSE

Das Programm Gaspreis soll noch weitergeführt werden. Der Tarif eines Gasversorgungsunternehmens ist wie folgt aufgebaut.

Tarif	Verbrauch in Kubikmeter	Grundgebühr	Preis pro Kubikmeter
I	kleiner als 4000	20.00 DM	0.3824 DM
II	4000 und mehr	27.50 DM	0.34655 DM

Die Tarifgestaltung soll in das Programm Gaspreis aufgenommen werden.
Welcher Tarif anzuwenden ist, richtet sich nach der Höhe des Verbrauchs. Ist der Verbrauch kleiner als 4000 Kubikmeter (cbm), wird nach Tarif I abgerechnet. Beträgt der Verbrauch 4000 cbm oder mehr, wird der Tarif II zu Grunde gelegt:
Die Situation kann folgendermaßen notiert werden:

WENN Verbrauch < 4000 DANN Preis := 20 + Verbrauch * 0.3824

beziehungsweise

WENN Verbrauch >= 4000 DANN Preis := 27.5 + Verbrauch * 0.34655

Diese beiden Auswahlanweisungen können in einer zweiseitigen Auswahl zusammengefaßt werden, da sich die beiden Abrechnungsbedingungen gegenseitig ausschließen.

WENN Verbrauch < 4000
    DANN Preis := 20 + Verbrauch * 0.3824
    SONST Preis := 27.5 + Verbrauch * 0.34655

IF-THEN-ELSE-Anweisung

Eine solche WENN-DANN-SONST-Konstruktion lautet in Pascal:

```
if   Verbrauch < 4000
     then Preis := 20 + Verbrauch * 0.3824
     else Preis := 27.5 + Verbrauch * 0.34655;
```

Die Schlüsselwörter lauten **if**, **then** und **else**. Hinter if wird der Bedingungsausdruck angegeben, von dessen Wahrheitswert (true oder false) es abhängt, welche Anweisung ausgeführt wird. Hat der Bedingungsausdruck den Wert **true**, so wird die Anweisung hinter **then** ausgeführt. Ist sein Wahrheitswert dagegen **false**, ist die Bedingung also nicht erfüllt, so wird die Anweisung hinter **else** bearbeitet.
Das Programm Gaspreis lautet dann unter Einbeziehung des Tarifgefüges:

```
program Gaspreis;
uses  crt;                                              (* ab Version 4 *)
const cbmPreis1 = 0.3824; cbmPreis2 = 0.34655;
      GrdGeb1   = 20;     GrdGeb2   = 27.5;
var AltZaehler, NeuZaehler, Verbrauch, Preis: real;

begin
  ClrScr;
  write ('Eingabe alter Zählerstand: '); readln (AltZaehler);
  write ('Eingabe neuer Zählerstand: '); readln (NeuZaehler);
  if NeuZaehler < AltZaehler
    then NeuZaehler := NeuZaehler + 100000.0;
  Verbrauch := NeuZaehler - AltZaehler;
  if Verbrauch < 4000
    then Preis := GrdGeb1 + Verbrauch * cbmPreis1
    else Preis := GrdGeb2 + Verbrauch * cbmPreis2;
 writeln('Der Gaspreis beträgt ',Preis:8:2);
end.
```

Erweitern Sie das Programm Gaspreis um die IF-THEN-ELSE-Anweisung, und testen Sie es mit folgenden Werten:

Beispiel 1	Beispiel 2
Eingabewerte	Eingabewerte
alter Zählerstand: 98500 neuer Zählerstand: 300	alter Zählerstand: 5000 neuer Zählerstand: 12000
Ausgabe	Ausgabe
Der Gaspreis beträgt 708.32	Der Gaspreis beträgt 2453.35

Syntax
Die allgemeine Form der IF-THEN-ELSE-Anweisung lautet:

if Bedingungsausdruck **then** Anweisung **else** Anweisung

Innerhalb der IF-Anweisung darf kein Semikolon stehen, also auch nicht vor then und else. Eine IF-Anweisung wird nämlich vom Compiler als eine einzige Anweisung angesehen, auch wenn innerhalb der IF-Anweisung mehrere Anweisungen enthalten sind.

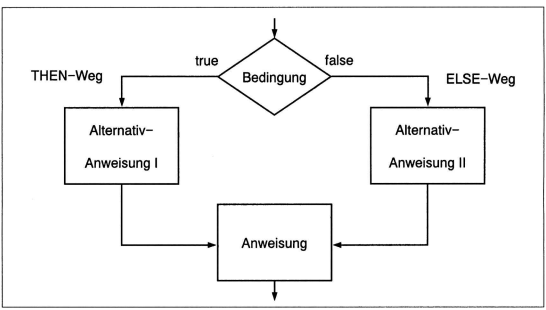
Vor then und else darf kein Semikolon stehen!

Folgen hinter then beziehungsweise else mehrere Anweisungen, so sind diese Anweisungen durch begin und end zu einer Verbundanweisung zusammenzufassen.

```
if  Bedingungsausdruck
    then begin
            Anweisung;
            Anweisung;
            ...
            end
    else begin
            Anweisung;
            Anweisung;
            ...
            end
```

Ablaufsteuerung
Die nachstehenden graphischen Darstellungen der zweiseitigen Auswahl in Form des Programmablaufplans sowie des Struktogramms

zeigen deutlich, daß die Anweisungen auf dem THEN-Weg und ELSE-Weg alternativ in Abhängigkeit vom Wahrheitswert des Bedingungsausdrucks ausgeführt werden.
Darstellung der einfachen Auswahl in Form eines Programmablaufplans:



Zweiseitige Auswahl in Form eines Struktogramms:

Bedingungsausdruck	
ja (true)	nein (false)
Alternative I	Alternative II
Anweisung	

Verschachtelte IF-Anweisung

In diesem Abschnitt soll eine komplette Lösung zu einem Problem entwickelt werden, das Ihnen in der Mathematik vielleicht einige Probleme bereitet hat: Lösung von quadratischen Gleichungen der Form

$$ax^2 + bx + c = 0$$

An diesem Beispiel wird aufgezeigt, wie ein komplexes Problem im Wege der **schrittweisen Verfeinerung** gelöst wird.

***Programmbeispiel* Quadratische Gleichung**

Die Lösungsformel ist Ihnen sicherlich bekannt:

$$x1,2 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Problemanalyse

Wenn die obige Lösungsformel im Programm verwendet werden soll, so sind zwei Bedingungen zu beachten. Zum einen darf der Wert für a nicht null sein. Sonst würde es zu einer Division durch null kommen. Die ist aber nicht zulässig. Zum anderen darf der Wert unter der Wurzel, die Diskriminante, nicht negativ sein. Aus negativen Zahlen kann nämlich keine reellwertige Wurzel gezogen werden.
Ist der Wert unter der Wurzel gleich null, so gibt es nur eine Lösung. Ist der Wert größer null, so gibt es zwei Lösungen.

Datenanalyse

Ausgabe-daten:	x1: real x2: real
Eingabe-daten:	a: real b: real c: real

Beziehungen zwischen Eingabe- und Ausgabedaten: Wie in der Problemanalyse dargelegt, hängt es von dem Wert unter der Wurzel, der sogenannten Diskriminanten, ab, wie viele Lösungen die quadratische Gleichung hat. Die Diskriminante bestimmt sich aus:

Diskriminante := $b^2 - 4ac$

Für Diskriminante > 0:

```
x1 := (–b + √Diskriminante) / (2*a)
x2 := (–b – √Diskriminante) / (2*a);
```

Für Diskriminante = 0:

x1 := –b / (2*a)

Für Diskriminante < 0:

Keine Lösung

Damit sind alle Ausgabedaten auf Eingabedaten zurückgeführt.

Algorithmus

Bei dem Aufbau des Algorithmus wird die Methode der

schrittweisen Verfeinerung

angewendet. Zunächst wird der Algorithmus nur in groben Zügen formuliert und dann unter Berücksichtigung der Verarbeitungsbedingungen Schritt für Schritt verfeinert:

1. Schritt: Algorithmisches Grundgerüst

```
Algorithmus Quadratische_Gleichung
Beginn
Eingabe der Werte für a, b, c
Berechnung der Diskriminanten
Berechnung von x1
Berechnung von x2
Ausgabe von x1 und x2
Ende
```

In dieses grobe algorithmische Gerüst werden nun die verschiedenen Verarbeitungsbedingungen einbezogen.

Erste Verfeinerung: Wenn der Wert für a gleich null ist, liegt keine quadratische Gleichung vor. Es brauchen dann auch die übrigen Berechnungen nicht vorgenommen zu werden.

```
Algorithmus Quadratische_Gleichung
Beginn
Eingabe der Werte für a, b, c
WENN a = 0
    DANN drucke (Keine quadratische Gleichung)
    SONST
```

```
        Berechnung der Diskriminanten
        Berechnung von x1
        Berechnung von x2
        Ausgabe von x1 und x2
Ende
```

Zweite Verfeinerung: Wenn die Diskriminante kleiner null ist, gibt es keine reelle Lösung. Dann brauchen x1 und x2 nicht berechnet und ausgegeben zu werden.

```
Algorithmus Quadratische_Gleichung
Beginn
Eingabe der Werte für a, b, c
WENN a = 0
   DANN drucke (Keine quadratische Gleichung)
   SONST
      Berechnung der Diskriminanten
      WENN Diskriminante < 0
         DANN drucke (Keine reelle Lösung)
         SONST
            Berechnung von x1
            Berechnung von x2
            Ausgabe von x1 und x2
Ende
```

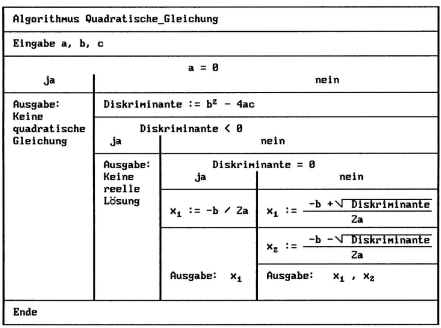
Dritte Verfeinerung: Wenn die Diskriminante gleich null ist, gibt es nur eine Lösung. Sie ist dann zu berechnen und auszudrucken. Die Berechnung und Ausgabe von x1 und x2 ist also nur dann vorzunehmen, wenn die Diskriminante größer null ist.

```
Algorithmus Quadratische_Gleichung
Beginn
Eingabe der Werte für a, b, c
WENN a = 0
   DANN Ausgabe (Keine quadratische Gleichung)
   SONST Berechnung der Diskriminanten
      WENN Diskriminante < 0
         DANN Ausgabe (Keine reelle Lösung)
         SONST WENN Diskriminante = 0
```

DANN Berechnung von x1
Ausgabe (x1)
SONST Berechnung von x1
Berechnung von x2
Ausgabe (x1, x2)

Ende

In drei Schritten wurde der Grob-Algorithmus so weit verfeinert, daß sämtliche Verarbeitungsbedingungen in WENN-DANN-Anweisungen berücksichtigt sind.
Hier wurde zur Entwicklung des Algorithmus der Pseudocode verwendet. Natürlich hätte man auch eine graphische Darstellungsform wie beispielsweise das Struktogramm wählen können. Aus dem nachstehend abgebildeten Struktogramm ist die Schachtelung von Auswahlanweisungen deutlich zu erkennen.



Erstellung des Programms

Bevor der Algorithmus in ein Programm umgesetzt werden kann, muß geklärt werden, wie in Pascal das Quadrat und die Wurzel einer Zahl berechnet werden.
Zur Berechnung der Quadratwurzel aus einer Zahl steht die Standardfunktion **sqrt** (squareroot; deutsch: Quadratwurzel) zur Verfügung.

Beispiel: sqrt(16) = 4

Der Wert beziehungsweise die Variable, aus der die Wurzel zu ziehen ist, wird in runden Klammern hinter dem Funktionsnamen **sqrt** angegeben. Standardfunktionen stellen wie Standardprozeduren kleine Programme dar, die über den Funktionsnamen aufgerufen werden. Zur Berechnung des Quadrats einer Zahl kann man die Standardfunktion **sqr** (square; deutsch: Quadrat) verwenden.

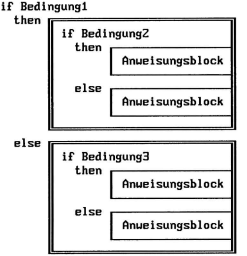
Beispiel: sqr(4) = 16

Der Wert beziehungsweise die Variable, die zu quadrieren ist, wird in runden Klammern hinter dem Funktionsnamen **sqr** angegeben.

Das Programm lautet dann:

```
program Quadratische_Gleichung;
uses crt;                                          (* ab Version 4 *)
var a, b, c : real;
    x1, x2, diskriminante: real;

begin
  ClrScr;
  writeln ('Mit diesem Programm können quadratische Gleichungen');
  writeln ('der Form a * x^2 + b * x + c = 0 gelöst werden');
  writeln;
  writeln ('Eingabe der Werte für');
  write ('  a: '); readln (a);
  write ('  b: '); readln (b);
  write ('  c: '); readln (c);
  writeln;
  if a = 0
    then writeln ('Keine quadratische Gleichung!')
    else begin
      diskriminante := sqr(b) - 4*a*c;
      if diskriminante < 0
        then writeln ('Keine reelle Lösung!')
        else if diskriminante = 0
               then begin
                 writeln ('Die Gleichung hat nur eine Lösung');
                 x1 := -b / (2*a);
                 writeln ('x = ',x1:10:4);
               end
               else begin
                 writeln ('Die Gleichung hat zwei Lösungen');
                 x1 := (-b + sqrt(diskriminante))/ (2*a);
                 x2 := (-b - sqrt(diskriminante))/ (2*a);
                 writeln('x1 = ',x1:10:4);
                 writeln('x2 = ',x2:10:4);
               end;
      end;
end.
```

Wie aus dem obigen Programmbeispiel Quadratische_Gleichung zu ersehen ist, können innerhalb einer IF-THEN-ELSE-Anweisung weitere Auswahlanweisungen programmiert werden. Dabei ist jedoch zu beachten, daß sich die Auswahlanweisungen nicht gegenseitig überlappen dürfen. Eine nachgeordnete Auswahlanweisung muß ganz in der übergeordneten eingeschachtelt sein.



Leeranweisung

Bei Programmen mit vielen verschachtelten IF-Anweisungen stellt sich die Frage, zu welcher IF-Anweisung ein else gehört. Optisch ist das in dem obigen Programm wohl zu erkennen, weil entsprechende Einrückungen vorgenommen worden sind. Den Compiler interessieren aber nicht die Einrückungen. Er verfährt streng nach der Regel, daß ein else sich auf das jeweils unmittelbar voraufgehende if bezieht. Diese Regel kann zusammen mit der Regel, daß vor else kein Semikolon stehen darf, aber auch zu Komplikationen führen!

***Programmbeispiel* Kundenrabatt**

In einem Unternehmen gilt folgende Rabattregelung:

Kundennummer	gewährter Rabattsatz
kleiner 3000	12 %
3000 bis 5000	0 %
größer 5000	10 %

Mit Hilfe des folgenden Programms soll nach der Eingabe der Kundennummer der entsprechende Rabattsatz berechnet werden.

```
0  program Kundenrabatt;
1  var Kdnr, Rabsatz: integer;
2  begin
3   write ('Eingabe Kundennummer '); readln (Kdnr);
4   Rabsatz := 0;
5   if Kdnr <= 5000
6      then if Kdnr < 3000 then  Rabsatz := 12
7      else Rabsatz := 10;
8   writeln ('Rabattsatz ',Rabsatz:6);
9  end.
```

Der Programmlauf erbringt folgende Ergebnisse:

Eingegebene Kundennummer	4000	6000	2000
Ausgegebener Rabattsatz	10	0	12
Richtiger Rabattsatz	0	10	12

Die Ergebnisse sind mit Ausnahme des letzten falsch. Somit ist das Programm fehlerhaft, auch wenn es optisch richtig aussieht. Der Grund dafür: Das else in Zeile 7 wird gemäß der Regel auf das unmittelbar in Zeile 6 vorausgehende if Kdnr < 3000 bezogen.
Um das else dem Kdnr <= 5000 in Zeile 5 zuordnen zu können, muß man sich eines «Tricks» bedienen. Man schiebt hinter then Rabsatz := 12 noch ein else mit einer Leeranweisung ein. Das bedeutet, daß hinter dem else nichts mehr angegeben wird.

```
0  program Kundenrabatt;
1  var Kdnr, Rabsatz: integer;
2  begin
3   write ('Eingabe Kundennummer '); readln (Kdnr);
4   Rabsatz := 0;
5   if Kdnr <= 5000
6     then if Kdnr < 3000
           then  Rabsatz := 12
7          else  (* Leeranweisung *)
8     else Rabsatz := 10;
9   writeln ('Rabattsatz ',Rabsatz:6);
10 end.
```

Der Programmtest bringt folgende Ergebnisse:

Eingegebene Kundennummer	4000	6000	2000
Ausgegebener Rabattsatz	0	10	12
Richtiger Rabattsatz	0	10	12

Das Programm arbeitet nun fehlerfrei. Durch das Einschieben einer leeren ELSE-Anweisung (Zeile 7) nach der THEN-Anweisung wird erreicht, daß die Anweisung

```
else Rabsatz := 10 (Zeile 8)
```

sich auf

```
if Kdnr <= 5000 (Zeile 6)
```

bezieht.

Kommentar
In der Zeile 7 des obigen Programms wurde mit

```
(* Leeranweisung *)
```

ein Kommentar verwendet. Ein Kommentar besteht aus einer beliebigen Folge von Zeichen, die in runde Klammern mit Stern oder in geschweifte Klammern eingeschlossen wird.

```
{ Dies ist ein Kommentar }
```

oder

```
(* Dies ist ein Kommentar *)
```

Ein Kommentar wirkt sich in keiner Weise auf den Ablauf des Programms aus. Er dient lediglich dazu, Programmschritte für den Leser des Programmtextes zu erläutern.

4.2.1 Zusammenfassung

- Die zweiseitige Auswahl lautet:
  if Bedingungsausdruck
  then AnweisungA
  else AnweisungB
- AnweisungA wird bearbeitet, wenn der Bedingungsausdruck wahr ist. Ist der Bedingungsausdruck falsch, wird AnweisungB ausgeführt.

- Die IF-Anweisung gilt als eine Anweisung. Deshalb darf vor then und vor else kein Semikolon stehen.
- Besteht der THEN-Zweig beziehungsweise der ELSE-Zweig aus mehreren Anweisungen, so sind sie durch begin und end zu einer Verbundanweisung zusammenzufassen.
- Zur Erläuterung einzelner Programmschritte kann der Programmierer Kommentare in das Programm einfügen. Kommentare werden in runden Klammern mit Stern (* ... *) oder in geschweiften Klammern { ... } geschrieben.

4.2.2 Übungen

1. Aufgabe

Gegeben ist folgendes Programm:

```
program test;
var a, b, c, d, e: real;
begin
  write ('Eingabe von a '); readln (a);
  write ('           b '); readln (b);
  write ('           c '); readln (c);
  d := 0; e := 0;
  if a < b
    then d := a + b;
  if a > c
    then if b > c
           then d := a * b
           else d := a * c
    else e := a + b + c;
  writeln ('d ',d:5:2,'e ',e:5:2);
end.
```

Welche Werte bilden sich in den Variablen d und e?

Eingabe in die Variablen			Ergebnisse	
a	b	c	d	e
7	3	4		
–3	11	46		
9	–5	5		
12	4	2		

2. Aufgabe

Der Staat erhebt Umsatzsteuer, auch Mehrwertsteuer (MWST) genannt. Sie beträgt in der Regel 15 % des Warenwerts. Für einige Warengruppen, wie zum Beispiel Lebensmittel, werden nur 7 % erhoben.

Bei einer Rechnungsschreibung wird außer dem Nettopreis ein Buchstabe als Mehrwertsteuerkennzeichen eingegeben. Ist der Buchstabe gleich «I», sind 7 % Mehrwertsteuer zu berechnen. Wurde ein anderer Buchstabe oder nur die [↵]-Taste gedrückt, so sind 15 % Mehrwertsteuer in Rechnung zu stellen.

	Beispiel 1	**Beispiel 2**
Eingabe	Nettobetrag: 100 MWST-Kennzeichen: I [↵]	Nettobetrag: 100 MWST-Kennzeichen: [↵]
Ausgabe	Umsatzsteuer: 7.00 Bruttopreis: 107.00	Umsatzsteuer: 15.00 Bruttopreis: 115.00

Ergänzen Sie folgenden Algorithmus, und schreiben Sie das Programm!

Algorithmus MehrwertsteuerBerechnung	
Konstante MWST7 = 7 MWST15 = 15	
Eingabe Nettopreis	
Eingabe MWST_Kennzeichen	
MWST_Kennzeichen = ... ja	nein
MWST := Nettopreis * MWST7/100	MWST := Nettopreis * MWST15/100
Bruttopreis := Nettopreis + MWST	
Ausgabe MWST	
Ausgabe Bruttopreis	
Ende	

3. Aufgabe

In einem Unternehmen beträgt laut Betriebsvereinbarung die durchschnittliche monatliche Arbeitszeit 175 Stunden. Leistet ein Mitarbeiter mehr als 175 Stunden, werden die Überstunden mit einem Aufschlag von 10 % vergütet.

Berechnungsbeispiele:

	Beispiel 1		**Beispiel 2**	
Ein-	Arbeitsstunden:	175	Arbeitsstunden:	183
gabe	Stundenlohn:	15.00	Stundenlohn:	15.00
Aus-	Gesamtstundenlohn:	2625.00	Gesamtstundenlohn:	2745.00
gabe	Überstundenzuschlag:	0.00	Überstundenzuschlag:	12.00
	Bruttolohn:	2625.00	Bruttolohn:	2757.00

a) Führen Sie die Problem- und Datenanalyse durch.
b) Entwickeln Sie den Algorithmus in Form eines Struktogramms.
c) Erstellen Sie das Programm, und testen Sie es.

4.2.3 Lösungen

1. Aufgabe

Eingabe in die Variablen			**Ergebnisse**	
a	b	c	d	e
7	3	4	28	0
–3	11	46	8	54
9	–5	5	45	0
12	4	2	48	0

2. Aufgabe

Algorithmus MehrwertsteuerBerechnung	
Konstante MWST7 = 7 MWST15 = 15	
Eingabe Nettopreis	
Eingabe MWST_Kennzeichen	
MWST_Kennzeichen = 'l'	
ja	nein
MWST := Nettopreis * MWST7/100	MWST := Nettopreis * MWST15/100
Bruttopreis := Nettopreis + MWST	
Ausgabe MWST	
Ausgabe Bruttopreis	
Ende	

Das Programm lautet:

```
program MehrwertsteuerBerechnung;
uses  crt;                                        (* ab Version 4 *)
const MWST7 = 7; MWST15 = 15;
var Nettopreis, MWST, Bruttopreis: real;
    MWST_Kennzeichen: char;
begin
  ClrScr;
  write ('Eingabe Nettopreis: '); readln (Nettopreis);
  write ('MWST-Kennzeichen: ');
  readln (MWST_Kennzeichen);
  if MWST_Kennzeichen = 'l'
    then MWST := Nettopreis * MWST7/100
    else MWST := Nettopreis * MWST15/100;
  Bruttopreis := Nettopreis + MWST;
  writeln ('Umsatzsteuer:',MWST:8:2);
  writeln ('Bruttopreis :',Bruttopreis:8:2);
end.
```

3. Aufgabe

```
program bruttolohnberechnung;
uses  crt;                                              (* ab Version 4 *)
const SollStunden = 175;
var Arbeitsstunden, Stundenlohn, Ueberstunden : real;
    Gesamtstundenlohn, Ueberstundenzuschlag, Bruttolohn: real;

begin
  ClrScr;
  writeln ('Vereinfachte Bruttolohnberechung');
  writeln ('*****************************');
  writeln; writeln;
  write ('Anzahl der Arbeitsstunden: '); readln (Arbeitsstunden);
  write ('Stundenlohn in DM        : '); readln (Stundenlohn);
  Gesamtstundenlohn := Arbeitsstunden * Stundenlohn;
  if Arbeitsstunden > SollStunden
    then begin
           Ueberstunden :=  Arbeitsstunden - SollStunden;
           Ueberstundenzuschlag := Ueberstunden*Stundenlohn/10;
           Bruttolohn := Gesamtstundenlohn + Ueberstundenzuschlag;
         end
    else begin
           Ueberstundenzuschlag := 0;
           Bruttolohn := Gesamtstundenlohn;
      end;
  writeln;
  writeln ('Gesamtstundenlohn   :',Gesamtstundenlohn:10:2);
  writeln ('Überstundenzuschlag:',Ueberstundenzuschlag:10:2);
  writeln;
  writeln ('Bruttolohn          :',Bruttolohn:10:2);
end.
```

4.3 Mehrseitige Auswahl: CASE

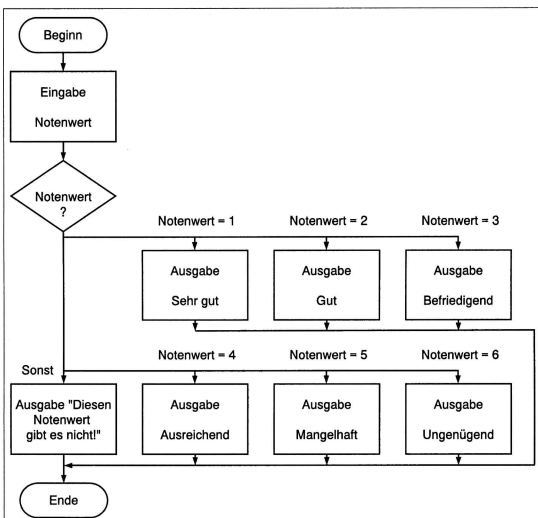
Durch die mehrseitige Auswahl können Sie unter bestimmten Umständen die Konstruktion geschachtelter IF-Anweisungen umgehen und dadurch das Programm einfacher gestalten.

Programmbeispiel **Notenskala**
Es ist ein Programm zu entwickeln, das die Notenwerte von 1 bis 6 in die Notenbezeichnungen Sehr gut, Gut, ..., Ungenügend umsetzt. Der Notenwert wird eingegeben. Der Computer zeigt die entsprechende Notenbezeichnung an.
Der Algorithmus zu diesem Problem lautet im Pseudocode:

Beginn
Eingabe Notenwert

Falls Notenwert gleich
    1: Schreibe Sehr gut
    2: Schreibe Gut
    3: Schreibe Befriedigend
    4: Schreibe Ausreichend
    5: Schreibe Mangelhaft
    6: Schreibe Ungenügend
    Sonst Schreibe Diesen Notenwert gibt es nicht
Ende

In Form eines Programmablaufplans lautet der Algorithmus:



In Form eines Struktogramms wird der Algorithmus zur Notenwertübersetzung wie folgt geschrieben:

Algorithmus Notenwertübersetzung						
Eingabe Notenwert						
Falls Notenwert						
1	2	3	4	5	6	Sonst
Schreibe Sehr gut	Schreibe Gut	Schreibe Befriedigend	Schreibe Ausreichend	Schreibe Mangelhaft	Schreibe Ungenügend	Schreibe Unzulässiger Wert
Ende						

Die mehrseitige Auswahl wird in Pascal durch die CASE-Anweisung verwirklicht. Der Kopf der CASE-Anweisung lautet in dem obigen Beispiel:

case Notenwert **of**

Vom Inhalt der Variablen Notenwert hängt es nun ab, welche der folgenden Anweisungen ausgeführt wird.

```
1: writeln ('Sehr gut');
2: writeln ('Gut');
3: writeln ('Befriedigend');
4: writeln ('Ausreichend');
5: writeln ('Mangelhaft');
6: writeln ('Ungenügend');
else writeln('Diesen Notenwert gibt es nicht!');
end (* von case *)
```

Ist der Inhalt der Variablen Notenwert gleich 2, so wird die Anweisung ausgeführt, vor der eine 2 steht. Hier würde das Wort Gut geschrieben. Alle anderen Anweisungen werden dann nicht beachtet. Wenn in Notenwert die Zahl 8 enthalten ist, wird die Anweisung ausgeführt, die hinter else steht. Auf dem Bildschirm würde dann angezeigt, daß es diesen Notenwert nicht gibt.
Das end zeigt das Ende der CASE-Struktur an. Da zu diesem end kein ausdrückliches begin gehört, setzt man gern in Kommentarklammer (* von case *) dahinter, um es beim Lesen des Programms dem case zuordnen zu können.
Das vollständige Programm lautet:

```
program Notenskala;
uses crt;                                          (* ab Version 4 *)
var Notenwert: integer;
begin
  ClrScr;
  writeln('Mit diesem Programm werden Notenwerte in Notenbezeichnungen');
  writeln('umgesetzt');
  writeln; writeln;
  write ('Eingabe des Notenwertes '); readln (Notenwert);
  case Notenwert of
    1:   writeln ('Sehr gut');
    2:   writeln ('Gut');
    3:   writeln ('Befriedigend');
    4:   writeln ('Ausreichend');
    5:   writeln ('Mangelhaft');
    6:   writeln ('Ungenügend');
    else writeln ('Diesen Notenwert gibt es nicht!');
  end;  (* von case *)
end.
```

Geben Sie das Programm ein, und testen Sie es.

CASE-Anweisung
Allgemein lautet die CASE-Anweisung

```
case Selektor of
    Auswahlmarke1: Anweisung1;
    Auswahlmarke2: Anweisung2;
    ...
    else Sonst_Anweisung;
end
```

Die mehrseitige Auswahl wird mit dem Schlüsselwort **case** eingeleitet. Dahinter folgt der Selektor. In dem obigen Programmbeispiel Notenskala ist der Selektor eine Integer-Variable. Aber auch ein arithmetischer Ausdruck kann als Selektor verwendet werden.

Beispiel:

```
case Ganzzahl+2 of
```

Entscheidend ist, daß der Datentyp des Selektors ordinal (das heißt abzählbar) sein muß. Der Selektor kann also vom Typ Integer oder Char sein, aber niemals Real.
Auf den Selektor folgt das Schlüsselwort **of**.
Hinter of werden die **Auswahlmarken** mit **Doppelpunkt** notiert. Auf

den Doppelpunkt folgt die Anweisung. Gehören zu einer Auswahlmarke mehrere Anweisungen, so werden sie zu einer Verbundanweisung mit begin und end zusammengefaßt.
Die Auswahlmarken müssen Konstanten sein. Ihr Datentyp muß mit dem Datentyp des Selektors übereinstimmen. Wenn die Selektor-Variable Buchstabe vom Typ Char (= Zeichen) ist, ist folgende CASE-Struktur falsch:

```
case Buchstabe of
     1: Anweisung1;
     2: Anweisung2;
     3: Anweisung3;
end; (* von case*)
```

Richtig ist dagegen folgender Programmausschnitt, da die Auswahlmarken aus Zeichenkonstanten bestehen, also vom selben Datentyp wie der Auswahlausdruck sind:

```
case Buchstabe of      oder    case Buchstabe of
'a': Anweisung1;               '1': Anweisung1;
'b': Anweisung2;               '2': Anweisung2;
'c': Anweisung3;               '3': Anweisung3;
end; (*von case*)              end; (*von case*)
```

Als Besonderheit weist Turbo-Pascal die Marke else auf. Hinter else steht die Anweisung, die auszuführen ist, wenn keine der Auswahlmarken mit dem Inhalt des Selektors übereinstimmt. Die else-Marke ist optional. Optional bedeutet, daß sie programmiert werden kann, wenn die Aufgabenstellung es erfordert. Sie kann aber auch fehlen, wenn sie nicht erforderlich ist.
Bei der Formulierung der Auswahlmarken hat der Programmierer mehrere Möglichkeiten. Sie werden an Hand von zwei Programmbeispielen im übernächsten Abschnitt dargestellt.
Die CASE-Anweisung wird immer durch ein end abgeschlossen.

Ablaufsteuerung
In einer CASE-Anweisung wird die Anweisung ausgeführt, deren Auswahlmarke mit dem Inhalt des Selektors übereinstimmt. Alle übrigen Anweisungen innerhalb der CASE-Anweisung werden dann übergangen.

Entspricht keine Auswahlmarke dem Wert des Selektors, so wird die Anweisung ausgeführt, vor der **else** steht. Fehlt ein ELSE-Zweig, so wird keine Anweisung der CASE-Struktur bearbeitet. Das Programm fährt dann mit der Anweisung fort, die dem end der CASE-Anweisung folgt.

Gestaltung von Auswahlmarken

Um die CASE-Anweisung möglichst problemnah programmieren zu können, gibt es verschiedene Möglichkeiten, die Auswahlmarken zu formulieren. Das wird an zwei Beispielen aufgezeigt.

***Programmbeispiel* Oberstufennoten**

In der gymnasialen Oberstufe werden nicht mehr die Notenwerte von 1 bis 6 vergeben. Die Notenwertskala reicht hier von 0 bis 15. Diesen Notenwerten sind folgende Notenbezeichnungen zugeordnet:

0	=	Ungenügend
1,2,3	=	Mangelhaft
4,5,6	=	Ausreichend
7,8,9	=	Befriedigend
10,11,12	=	Gut
13,14,15	=	Sehr Gut

Der Notenwert wird eingegeben. Der Computer soll die entsprechende Notenbezeichnung ausgeben.
Das Programm dazu lautet:

```
program Oberstufennoten;
var Notenwert:integer;
begin
  write ('Eingabe des Notenwertes '); readln (Notenwert);
  case Zensurenwert of
    15, 14, 13: writeln ('Sehr gut');
    12, 11, 10: writeln ('Gut');
     9,  8,  7: writeln ('Befriedigend');
     6,  5,  4: writeln ('Ausreichend');
     3,  2,  1: writeln ('Mangelhaft');
           0: writeln ('Ungenügend');
    else writeln ('Die Note gibt es nicht!');
  end;  (* von case *)
end.
```

Geben Sie das Programm ein, und testen Sie es.
Hier besteht die Auswahlmarke aus mehreren Konstanten, die jeweils durch Komma voneinander getrennt werden.

***Programmbeispiel* Auskunft**

In einem großen Finanzamt wird der Pförtner gefragt: «Wer ist mein Einkommensteuersachbearbeiter?» Der Pförtner fragt zurück: «Mit welchem Buchstaben beginnt Ihr Hausname?» Der Gefragte nennt den Buchstaben. Der Pförtner tippt den Buchstaben in den Computer ein. Am Bildschirm erscheinen der Name und die Zimmernummer des Steuersachbearbeiters.

Wegweiser

Buchstabe	Sachbearbeiter	Zimmer
A–C	Herr Adam	29
D–G	Frau Bonte	35
H–K	Frau Summen	45
L–P	Herr Bentler	17
Q–Z	Frau Sauer	43

Der Rechner in der Pförtnerloge arbeitet mit folgendem Programm:

```
program Auskunft;
uses crt;                                          (* ab Version 4 *)
var Anfangsbuchstabe: char;
begin
  ClrScr;
  write ('Eingabe des Anfangsbuchstabens: '); readln(Anfangsbuchstabe);
  writeln;
  write ('Der zuständige Sachbearbeiter ist ');
  case Anfangsbuchstabe of
    'a' .. 'c': writeln ('Herr Adam, Zimmer 29');
    'd' .. 'g': writeln ('Frau Bonte, Zimmer 35');
    'h' .. 'k': writeln ('Frau Sunnen, Zimmer 45');
    'l' .. 'p': writeln ('Herr Bentler, Zimmer 17');
    else        writeln ('Frau Sauer, Zimmer 43');
  end  (* von case *)
end.
```

In diesem Programm wurden die Konstanten, aus denen die Auswahlmarke besteht, nicht einzeln aufgezählt. Vielmehr wurden für die Auswahlmarken jeweils Bereiche (genauer: Unterbereiche) angegeben. Die Auswahlmarke

'l'..'p'

bedeutet beispielsweise, daß sie die Buchstaben

'l','m','n','o','p'

umfaßt.

Unterbereiche

Ein Unterbereich besteht aus endlich vielen aufeinanderfolgenden Elementen eines vordefinierten ordinalen Datentyps, zum Beispiel Integer-Zahlen oder Zeichen, niemals aber Real-Zahlen. Er wird festgelegt durch die Angabe

Untergrenze .. Obergrenze

Untergrenze und Obergrenze sind Konstanten des jeweiligen Datentyps, aus denen der Teilbereich gebildet wird. So stellt beispielsweise der Unterbereich

5 .. 30

einen Ausschnitt aus dem Wertebereich des Datentyps Integer dar. Wie auch dies letzte Beispiel zeigt, erleichtert die CASE-Struktur die Gestaltung von Programmen. CASE-Anweisungen können aber nur

dort eingesetzt werden, wo auf einfache Daten ordinalen Typs abgefragt wird. Hängt die Auswahl von Anweisungen ab vom Wert einer Real-Variablen, so muß dann doch wieder auf die IF-Anweisung zurückgegriffen werden.

4.3.1 Zusammenfassung

- Die mehrseitige Auswahl lautet:

```
case  Selektor of
        Auswahlmarke1: Anweisung_1;
        Auswahlmarke2: Anweisung_2;
        .
        .
        else Sonst_Anweisung;
        end (* von case *)
```

- Selektor kann eine Variable oder ein Ausdruck sein. Der Datentyp muß ein Ordinaltyp sein.
- Die Auswahlmarken sind Konstanten, die denselben Datentyp wie der Selektor haben müssen. Sie können auch in der Form eines Unterbereichs, zum Beispiel 1..5 oder 'A'..'K' angegeben werden.
- Es wird die Anweisung ausgeführt, deren Auswahlmarke mit dem Inhalt des Selektors übereinstimmt.
- Wenn keine Auswahlmarke mit dem Inhalt des Selektors übereinstimmt, wird die Anweisung hinter else ausgeführt. Fehlt der ELSE-Zweig, wird mit der Anweisung fortgefahren, die der CASE-Anweisung folgt.

4.3.2 Übungen

1. Welchen Syntaxfehler enthält folgendes Programm?

```
program Proben;
var x: real;
begin
  write ('Gib eine positive Zahl ein ');readln (x); writeln;
  write ('Die eingegebene Zahl liegt zwischen ');
  case x of
     0 ..   10: writeln ('   0 und   10');
    11 ..  100: writeln ('  11 und  100');
   101 .. 1000: writeln (' 101 und 1000');
    else        writeln ('1001 und unendlich');
  end  (* von case *)
end.
```

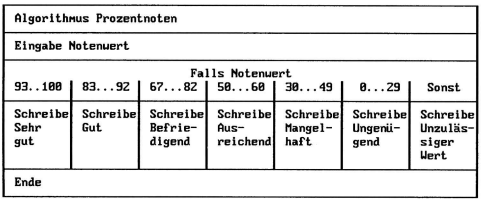
2. In einer CASE-Anweisung ist kein ELSE-Zweig programmiert worden. Was geschieht im Turbo-Pascal-System, wenn der Wert des Auswahlausdrucks nicht mit einer Auswahlmarke übereinstimmt?
3. Bei Berufsabschlußprüfungen werden die erbrachten Leistungen in Prozent bewertet. Den Prozentangaben werden dann Notenbezeichnungen zugeordnet. Dabei soll folgende Skala gelten:

Prozentbereich	Note
0–29	Ungenügend
30–49	Mangelhaft
50–66	Ausreichend
67–82	Befriedigend
83–92	Gut
93–100	Sehr Gut

Das Struktogramm und das Programm sind so zu gestalten, daß der erreichte Prozentsatz eingegeben wird. Der Computer soll die entsprechende Notenbezeichnung ausgeben. Wird eine Prozentangabe eingegeben, die außerhalb der obigen Prozentbereiche liegt, soll eine Fehlermeldung auf dem Bildschirm erscheinen.

4.3.3 Lösungen

1. Der Auswahlausdruck x ist eine Real-Variable. Sie ist somit nicht ordinalen Typs. Skalare, d. h. abzählbare Typen sind beispielsweise ganze Zahlen oder Zeichen.
2. Es wird keine Anweisung der CASE-Struktur bearbeitet. Das Programm setzt mit der Anweisung fort, die dem end, das das Ende der CASE-Anweisung anzeigt, folgt.
3. Das Struktogramm hat die Form:

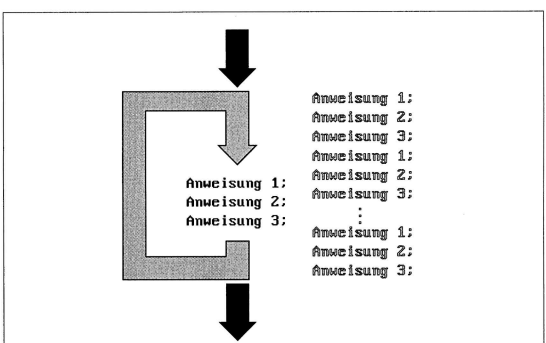


Das Programm hat die Form:

```
program Prozentnoten;
var Prozent: integer;
begin
  write ('Erreichte Prozentleistung '); readln (Prozent);
  case Prozent of
    93 ..100: writeln ('Sehr gut');
    83 .. 92: writeln ('Gut');
    67 .. 82: writeln ('Befriedigend');
    50 .. 66: writeln ('Ausreichend');
    30 .. 49: writeln ('Mangelhaft');
     0 .. 29: writeln ('Ungenügend');
    else      writeln ('Falsche Prozentangabe!');
  end  (* von case *)
end.
```

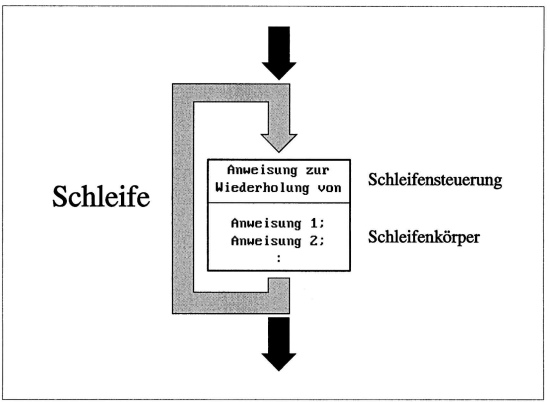
5 WIEDERHOLUNGS-ANWEISUNGEN

Bei vielen Aufgaben, die vom Computer zu bearbeiten sind, müssen Anweisungen wiederholt ausgeführt werden. Es wäre aber viel zu umständlich, die Anweisungen so oft ins Programm zu schreiben, wie sie zu wiederholen sind.



Durch Verwendung von Wiederholungsanweisungen (= Iterationsanweisungen) wird dieses Problem elegant gelöst. Die Folge von Anweisungen, die wiederholt ausgeführt werden muß, wird nur einmal geschrieben. Mittels einer Iterationsanweisung wird gesteuert, wie oft die Folge von Anweisungen bearbeitet wird.
Eine Iteration wird oft auch als Schleife bezeichnet. In der Iterationsanweisung ist die Schleifensteuerung festgelegt. Die Anweisun-

gen, die wiederholt ausgeführt werden sollen, stellen den Schleifenkörper dar. Das veranschaulicht auch die nachstehende Abbildung.



In Pascal gibt es drei verschiedene Wiederholungsanweisungen:

repeat – until
while – do
for – to – do bzw. for – downto – do

Diese drei Schleifenformen werden nachfolgend an einem Beispiel dargestellt:

***Programmbeispiel* Amerikanische Versteigerung**
Bei der amerikanischen Versteigerung beginnt das erste Gebot bei 1 DM. Der Bieter legt 1 DM in den «Pott». Der nächste Bieter erhöht auf 2 DM. Er zahlt 2 DM in den «Pott» usw. Es wird jedesmal um 1 DM erhöht, und der gebotene Betrag muß in den «Pott» bezahlt werden. Derjenige, der als letzter geboten hat, erhält den Zuschlag. Diese Art der Versteigerung macht Spaß und bringt auch eine Menge Geld ein. Überzeugen Sie sich selbst, daß in nur wenigen Minuten

eine schöne Summe zusammenkommt, auch wenn beispielsweise nur fünfzehnmal geboten wird.
Sie brauchen nun nicht erst eine amerikanische Versteigerung zu organisieren, um zu sehen, wie schnell der «Pott» anwächst. Das wird mittels der nachstehenden Programme simuliert, d. h. nachvollzogen. Mathematisch gesehen geht es darum, die Summe aller natürlichen Zahlen von 1 bis 15 zu bilden. Dabei sollen die Zwischensummen ebenfalls angezeigt werden.
Bildschirmausgabe:

```
Gezahlt:  1 DM        Im Pott:   1 DM
Gezahlt:  2 DM        Im Pott:   3 DM
Gezahlt:  3 DM        Im Pott:   6 DM
...                   ...
Gezahlt: 15 DM        Im Pott: 120 DM
Gesamteinnahme: 120 DM
```

5.1 Fußgesteuerte Wiederholung: REPEAT – UNTIL

Den Ablauf einer amerikanischen Versteigerung simuliert folgendes Programm und errechnet dabei die Gesamteinnahme. Dabei werden die oben dargelegten Spielbedingungen zugrunde gelegt. Die Versteigerung soll auf 15 Gebote beschränkt sein.

```
0  program Amerikanische_Versteigerung;
   uses crt;                                    (* ab Version 4 *)
1  var Betrag, Pott: integer;
2  begin
3    ClrScr;
4    Pott := 0;
5    Betrag := 1;
6    repeat
7      Pott := Pott + Betrag;
8      writeln ('Gezahlt:',Betrag:5,' DM   Im Pott:',Pott:5,' DM');
9      Betrag := Betrag + 1;
10   until Betrag > 15;
11   writeln ('Gesamteinnahme:',Pott:5,' DM');
12 end.
```

Wenn Sie das Programm eingeben und starten, können Sie sehen, wie rasch viel Geld eingespielt wird.

Erläuterungen zu dem Programm:
In Zeile 4 wird der «Pott» auf null gestellt. Wenn nämlich zu Beginn der Versteigerung zufällig noch Geld im Topf wäre, hätte man am Ende der Versteigerung mehr Geld im Pott als tatsächlich eingenommen.
In Zeile 5 wird das erste Gebot auf den Betrag von 1 (DM) festgelegt.
In Zeile 7 wird der Betrag in den Pott einbezahlt, und es wird festgestellt, wieviel Geld jetzt im Topf ist. Die Schreibweise mag mit

```
Pott := Pott + Betrag;
```

ein wenig seltsam erscheinen. Sie ist aber logisch, weil sich der neue Inhalt von Pott aus dem ergibt, was bereits im Pott war, zuzüglich des eingezahlten Betrags.
In Zeile 8 können Sie die Zahlung und den Pott-Inhalt kontrollieren. Der (gezahlte) Betrag und die Summe, die sich im Pott angesammelt hat, werden am Bildschirm angezeigt.
In der Anweisung in Zeile 9 wird die Regel berücksichtigt, daß der nächste (zu zahlende) Betrag um genau 1 (DM) höher liegen muß als der vorhergehende (gezahlte) Betrag.
In Zeile 10 wird kontrolliert, ob bereits 15 Spielrunden gelaufen sind. Das wird hier daran festgestellt, daß (der jetzt zu zahlende) Betrag 15 (DM) überschritten hat.
Wenn Betrag aber noch nicht größer 15 ist, wird die Anweisung ausgeführt, die auf das Schlüsselwort **repeat** in Zeile 6 folgt. Das ist die Anweisung in Zeile 7. Anschließend werden die Anweisungen in den Zeilen 8 und 9 bearbeitet. In Zeile 10 erfolgt wieder die Kontrolle, ob das Spielende erreicht ist.
Die Anweisungen in den Zeilen 7–9 werden also so lange in einer Schleife wiederholt durchgeführt, bis Betrag größer als 15 ist. Diese Anweisungen stellen also den Schleifenkörper dar.
Da die Abfrage auf Abbruch der Wiederholung am Fuß des Schleifenkörpers erfolgt, spricht man bei dieser Schleifenkonstruktion von einer fußgesteuerten Schleife.

REPEAT-Anweisung
Die fußgesteuerte Schleife wird in Pascal durch die REPEAT-Anweisung verwirklicht. Allgemein lautet sie:

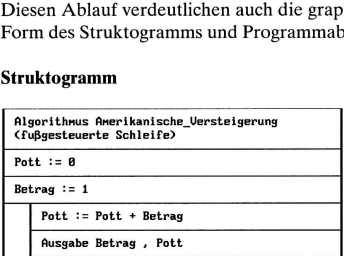
```
repeat
    Anweisung;
    Anweisung;
    ...
until Bedingungsausdruck
```

Das Schlüsselwort **repeat** (deutsch: wiederhole) kennzeichnet den Beginn der Schleife und steht vor der ersten zu wiederholenden Anweisung. Das Ende des Schleifenkörpers zeigt das Schlüsselwort **until** (deutsch: bis) an. Hinter until wird angegeben, unter welcher Voraussetzung bzw. Bedingung der Schleifenkörper nicht mehr bearbeitet werden soll. In dem Bedingungsausdruck hinter until wird also die Schleifen**austritts**bedingung beschrieben.

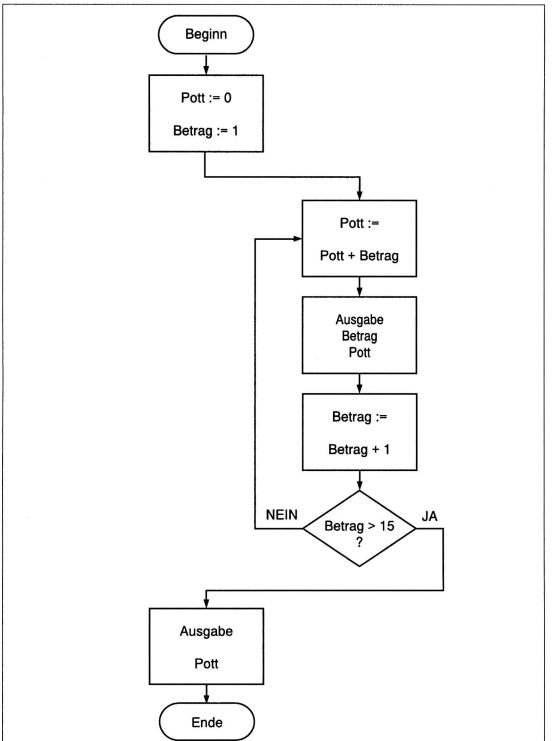
Ablaufsteuerung

Die Anweisungen des Schleifenkörpers werden immer mindestens einmal bearbeitet, da die Schleifenaustrittsbedingung erst am Ende des Schleifenkörpers geprüft wird. Ist die Bedingung nicht erfüllt, werden die Anweisungen des Schleifenkörpers wiederholt.
Ist die Bedingung erfüllt, weist sie also den Wahrheitswert true auf, fährt das Programm mit der Anweisung fort, die der REPEAT-Anweisung folgt.
Diesen Ablauf verdeutlichen auch die graphischen Darstellungen in Form des Struktogramms und Programmablaufplans.

Struktogramm



Programmablaufplan



5.1.1 Zusammenfassung

- Die Anweisungen, die zu wiederholen sind, bezeichnet man als Schleifenkörper.
- Bei einer fußgesteuerten Schleife wird am Ende des Schleifenkörpers über eine Bedingung geprüft, ob die Schleife erneut durchlaufen werden soll.
- Zur Konstruktion einer fußgesteuerten Schleife wird die REPEAT-Anweisung eingesetzt.
- Aufbau der REPEAT-Anweisung:
  repeat

Schleifen- körper

  until Bedingungsausdruck

- Hat der Bedingungsausdruck den Wahrheitswert false, wird die Schleife erneut bearbeitet. Die Schleife wird verlassen, wenn er den Wahrheitswert true aufweist.
- Bei einer REPEAT-Anweisung werden die Anweisungen des Schleifenkörpers mindestens einmal bearbeitet.
- Die Anweisung zur Aufsummierung lautet verallgemeinert:

  Summe := Summe + Wert

  Die Summenvariable ist vor Eintritt in die Aufsummierungsschleife auf null zu setzen.

5.1.2 Übungen

1. Aufgabe
Mit folgendem Programm soll die Summe der Zahlen

1 + 2 + 3 + 4 + ... + 97 + 98 + 99 + 100

gebildet werden. Ergänzen Sie die Schleifenaustrittsbedingung.

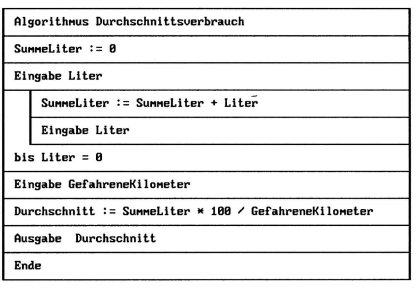
```
program Zahlen_Summierung;
var Zahl, Summe: integer;
begin
```

```
    Summe := 0;
    Zahl := 1;
    repeat
      Summe := Summe + Zahl;
      Zahl := Zahl + 1;
    until _____
    writeln('Die Summe der Zahlen beträgt',Summe:8);
end.
```

2. Aufgabe

Mit Hilfe eines Programms soll der durchschnittliche Kraftstoffverbrauch auf 100 km berechnet werden. Dazu werden die Literangaben der Tankbelege in den Rechner eingegeben, der die Werte sofort aufsummiert. Um dem Rechner anzuzeigen, daß sämtliche Literangaben erfaßt sind, wird nach der letzten Litereingabe eine Null eingetippt. Anschließend ist die Anzahl der gefahrenen Kilometer einzugeben.
Der Rechner multipliziert die Summe der Liter mit 100, dividiert das Produkt durch die Anzahl der gefahrenen Kilometer und zeigt das Ergebnis auf dem Bildschirm an.

Entwickeln Sie aus dem nachstehenden Struktogramm das Programm.



5.1.3 Lösungen

1. Aufgabe

```
program Zahlen_Summierung;
var Zahl, Summe: integer;
begin
  Summe := 0;
  Zahl := 1;
  repeat
    Summe := Summe + Zahl;
    Zahl := Zahl + 1;
  until Zahl > 100;
  writeln('Die Summe der Zahlen beträgt',Summe:8);
end.
```

2. Aufgabe

Das Programm lautet:

```
program Durchschnittsverbrauch;
uses crt;                                          (* ab Version 4 *)
var  Liter, SummeLiter, GefahreneKilometer, Durchschnitt: real;
begin
  ClrScr;
  SummeLiter := 0;
  write ('Liter: '); readln (Liter);
  repeat
    SummeLiter := SummeLiter + Liter;
    write ('Liter (Ende = 0): ');
    readln (Liter);
  until Liter = 0;
  write ('Wieviel Kilometer sind Sie gefahren? ');
  readln (GefahreneKilometer);
  Durchschnitt := SummeLiter * 100 / GefahreneKilometer;
  writeln ('Durchschnittlicher Literverbrauch', Durchschnitt:8:2);
end.
```

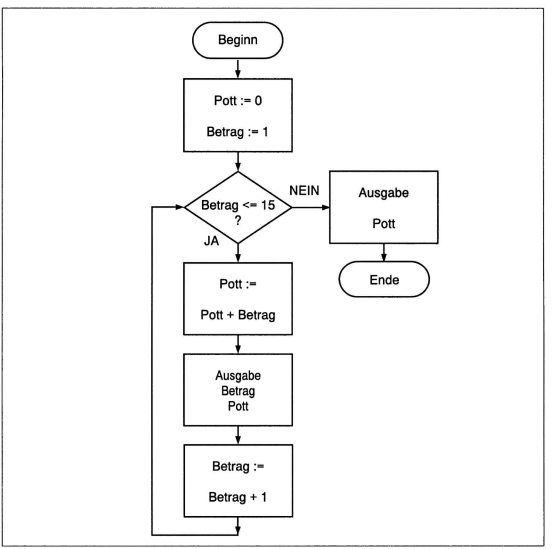
5.2 Kopfgesteuerte Wiederholung: WHILE – DO

Eine weitere Form der Schleifengestaltung besteht darin, daß vor dem Beginn des Schleifenkörpers in einem Bedingungsausdruck geprüft wird, ob die Anweisungen des Schleifenkörpers bearbeitet werden sollen. Man spricht dann von einer kopfgesteuerten Schleife. Der Algorithmus zu Amerikanische_Versteigerung wird in Form einer kopfgesteuerten Schleife graphisch wie folgt dargestellt.

Struktogramm

```
Algorithmus Amerikanische_Versteigerung
(kopfgesteuerte Schleife)
Pott := 0
Betrag := 1
Solange Betrag <= 15
    Pott := Pott + Betrag
    Ausgabe Betrag , Pott
    Betrag := Betrag + 1
Ende
```

Programmablaufplan



Unter Anwendung der kopfgesteuerten Schleife lautet das Programm «Amerikanische_Versteigerung»:

```
0  program Amerikanische_Versteigerung;
   uses crt;                                    (* ab Version 4 *)
1  var Betrag, Pott: integer;
2  begin
3    ClrScr;
4    Pott := 0;
5    Betrag := 1;
6    while Betrag <= 15 do
7      begin
8        Pott := Pott + Betrag;
9        writeln ('Gezahlt:',Betrag:5,' DM   In Pott:',Pott:5,' DM');
10       Betrag := Betrag + 1;
11     end;
12   writeln ('Gesamteinnahme:',Pott:5,' DM');
13 end.
```

Geben Sie das Programm ein, und testen Sie es. Es funktioniert genauso wie das obige Programm unter Verwendung der REPEAT-Anweisung.
Die kopfgesteuerte Schleife wird durch die WHILE-Anweisung realisiert. Ob die Anweisungen des Schleifenkörpers (Zeile 7 bis 11) ausgeführt werden, hängt ab von dem Bedingungsausdruck

```
Betrag <= 15
```

in der Zeile 6. Entsprechend dieser Bedingung wird der Schleifenkörper so lange wiederholt abgearbeitet, wie der Inhalt der Variablen Betrag kleiner oder gleich 15 ist. Ist der Inhalt von Betrag größer 15, fährt das Programm mit der Anweisung in Zeile 12 fort.

WHILE-Anweisung
Die allgemeine Form der WHILE-Anweisung lautet:

(a) Der Schleifenkörper besteht nur aus einer Anweisung.	(b) Der Schleifenkörper besteht aus mehreren Anweisungen.

while Bedingungsausdruck **do**
Anweisung
	while Bedingungsausdruck **do**
	begin
	Anweisung;
	Anweisung;
	...
	end

Das Schlüsselwort lautet **while** (deutsch: solange). Dahinter folgt ein Bedingungsausdruck. In dem Bedingungsausdruck wird angegeben, unter welcher Voraussetzung die Anweisung, die auf das **do** folgt, ausgeführt wird. Sind mehrere Anweisungen zu wiederholen, so werden diese Anweisungen durch begin und end zu einer Verbundanweisung zusammengefaßt.

Ablaufsteuerung

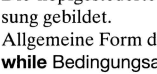
Im Unterschied zur REPEAT-Anweisung wird bei der WHILE-Anweisung vor Beginn des Schleifenkörpers geprüft, ob die Bedingung für den Eintritt in die Schleife überhaupt erfüllt ist.

Ist die Schleifeneintrittsbedingung erfüllt, hat sie also den Wahrheitswert true, werden die Anweisungen des Schleifenkörpers ausgeführt. Wenn die Schleifeneintrittsbedingung nicht erfüllt ist, werden die Anweisungen des Schleifenkörpers nicht bearbeitet. Das Programm fährt mit der Anweisung fort, die der WHILE-Anweisung folgt.

Bei der WHILE-Anweisung ist es deshalb möglich, daß die Anweisungen des Schleifenkörpers während eines Programmlaufs überhaupt nicht bearbeitet werden.

5.2.1 Zusammenfassung

- Bei einer kopfgesteuerten Schleife wird vor der Bearbeitung der Anweisungen des Schleifenkörpers geprüft, ob die Schleifeneintrittsbedingung erfüllt ist.
- Die kopfgesteuerte Schleife wird mit Hilfe der WHILE-Anweisung gebildet.
- Allgemeine Form der WHILE-Anweisung:
  while Bedingungsausdruck **do**

  Ist der Bedingungsausdruck nicht erfüllt, wird der Schleifenkörper übergangen und die Anweisung ausgeführt, die der WHILE-Anweisung folgt.

5.2.2 Übungen

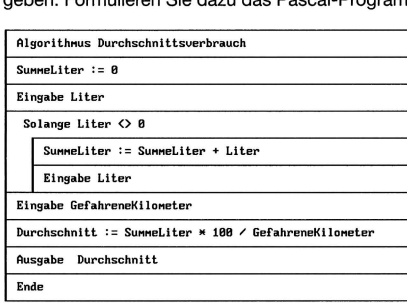
1. Aufgabe
Mit folgendem Programm soll die Summe der Zahlen

1 + 2 + 3 + 4 + ... + 97 + 98 + 99 + 100

gebildet werden. Ergänzen Sie die Schleifeneintrittsbedingung.

```
programm Zahlen_Summierung_2;
var Zahl, Summe: integer;
begin
   Summe := 0;
   Zahl := 1;
   while _____ do
     begin
     Summe := Summe + Zahl;
     Zahl := Zahl + 1;
     end;
   writeln('Die Summe der Zahlen beträgt',Summe:8);
end.
```

2. Aufgabe
Zu der Aufgabe 2 der Übung 5.2.1 ist folgender Algorithmus vorgegeben. Formulieren Sie dazu das Pascal-Programm!



5.2.3 Lösungen

1. Aufgabe

```
program Zahlen_Summierung_2;
var Zahl, Summe: integer;
begin
  Summe := 0;
  Zahl := 1;
  while Zahl <= 100 do
    begin
    Summe := Summe + Zahl;
    Zahl := Zahl + 1;
    end;
  writeln('Die Summe der Zahlen beträgt',Summe:8);
end.
```

2. Aufgabe

Das Programm lautet:

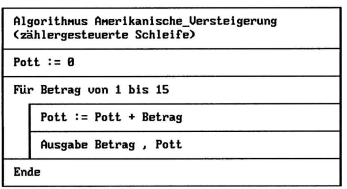
```
program Durchschnittsverbrauch;
uses crt;                                          (* ab Version 4 *)
var  Liter, SummeLiter, GefahreneKilometer, Durchschnitt: real;
begin
  ClrScr;
  SummeLiter := 0;
  write ('Liter: '); readln (Liter);
  while Liter <> 0 do
    begin
      SummeLiter := SummeLiter + Liter;
      write ('Liter (Ende = 0): ');
      readln (Liter);
    end;
  write ('Wieviel Kilometer sind Sie gefahren? ');
  readln (GefahreneKilometer);
  Durchschnitt := SummeLiter * 100 / GefahreneKilometer;
  writeln ('Durchschnittlicher Literverbrauch', Durchschnitt:8:2);
end.
```

5.3 Zählergesteuerte Wiederholung: FOR - TO - DO / FOR - DOWNTO - DO

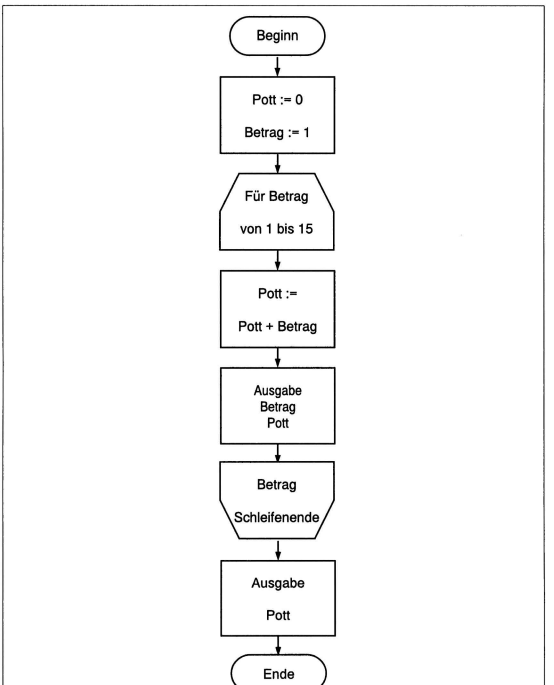
Wenn die Anzahl der erforderlichen Schleifendurchläufe schon vorher bekannt ist, kann als dritte Form der Wiederholungsanweisung die zählergesteuerte Schleife verwendet werden. In dem Programm-

beispiel Amerikanische_Versteigerung ist das der Fall. Laut Vereinbarung sollte ja der Ertrag aus 15 Spielrunden errechnet werden. Der Algorithmus zu Amerikanische_Versteigerung wird unter Verwendung der zählergesteuerten Schleife wie folgt dargestellt:

Struktogramm



Programmablaufplan



Die zählergesteuerte Schleife wird mittels der FOR-Anweisung umgesetzt.
Das Programm Amerikanische_Versteigerung lautet unter Verwendung der FOR-Anweisung:

```
 0  program Amerikanische_Versteigerung;
    uses crt;                                    (* ab Version 4 *)
 1  var Betrag, Pott: integer;
 2  begin
 3    ClrScr;
 4    Pott := 0;
 5    for Betrag := 1 to 15 do
 6      begin
 7        Pott := Pott + Betrag;
 8        writeln ('Gezahlt:',Betrag:5,' DM   In Pott:',Pott:5,' DM');
 9      end;
10    writeln ('Gesamteinnahme:',Pott:5,' DM');
11  end.
```

Geben Sie das Programm ein, und testen Sie es. Es funktioniert genauso wie die beiden vorherigen Programme.

Erläuterungen zu dem Programm:
Die Ausführung des Schleifenkörpers (Zeilen 6 bis 9) wird gesteuert über die Iterationsanweisung

```
for Betrag := 1 to 15 do
```

in Zeile 5. Man kann es übersetzen mit «für Betrag von 1 bis 15 tue ... ». Solange die Variable Betrag den Wert 15 nicht überschritten hat, werden die Anweisungen des Schleifenkörpers ausgeführt. Nach jedem Schleifendurchlauf wird der Inhalt der Variablen Betrag um 1 erhöht. Diese Erhöhung erfolgt automatisch.
Übersteigt die Variable Betrag den Wert 15, so wird die Schleife verlassen.
Das Programm fährt dann fort mit der Anweisung in Zeile 10.
Wenn Sie die beiden letzten Programmversionen vergleichen, so stellen Sie fest, daß das for-gesteuerte Programm um zwei Anweisungen kürzer ist als das while-gesteuerte. Die Anweisungen in dem while-gesteuerten Programm

```
 5 Betrag :=1
10 Betrag :=Betrag + 1
```

werden bei Verwendung der FOR-Anweisung nicht benötigt. Diese beiden Anweisungen werden von der FOR-Anweisung automatisch erledigt.

FOR-Anweisung

In allgemeiner Form lautet die FOR-Anweisung,

(a) wenn der Schleifenkörper aus nur einer Anweisung besteht:

```
for Variable := Anfangswert to Endwert do Anweisung
```

(b) wenn der Schleifenkörper aus mehreren Anweisungen besteht:

```
for Variable := Anfangswert to Endwert do
    begin
    Anweisung;
    Anweisung;
    ...
    end
```

Die FOR-Anweisung wird eingeleitet durch das Schlüsselwort **for**. Dahinter wird eine Variable angeführt, der ein Anfangswert zugewiesen wird. Die Variable wird auch **Kontrollvariable, Laufvariable** oder **Zählervariable** genannt. Hinter dem **to** wird ein Endwert angegeben. Auf das sich anschließende **do** folgt die Anweisung, die wiederholt werden soll.

Falls der Schleifenkörper aus mehreren Anweisungen besteht, so sind die Anweisungen mittels begin und end zu einer Verbundanweisung zusammenzufassen.

Die Kontrollvariable muß vom ordinalen Typ (= abzählbar) sein. Sie kann zum Beispiel eine Integer-Variable oder auch eine Char-Variable, aber niemals eine Real-Variable sein. Der Datentyp von Anfangswert und Endwert muß dem Datentyp der Kontrollvariablen entsprechen.

Beispiele:

zulässig: `for Zeichen := 'a' to 'z' do Anweisung`
In der Variablen Zeichen werden nacheinander die Kleinbuchstaben des Alphabets: a,b,c,...,z gebildet.

unzulässig: `for Zahl := 1.5 to 15.4 do Anweisung`

Andererseits darf die Angabe zu Anfangswert oder Endwert aus einem zu berechnenden Term bestehen, dessen Wert jedoch ordinalen Typs sein muß.

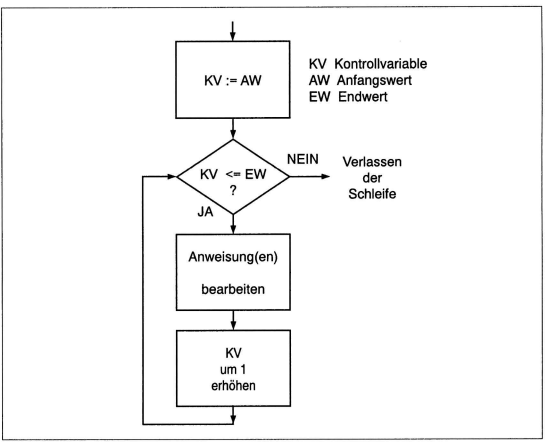
Beispiel:

```
Awert und Ewert sind vom Typ Integer.
for Zaehler := 2*Awert to Ewert + 3 do Anweisung
```

Ablaufsteuerung

Bei der ersten Bearbeitung der FOR-Anweisung wird der Kontrollvariablen der Anfangswert zugewiesen. Anschließend wird geprüft, ob der Inhalt der Kontrollvariablen kleiner oder gleich dem Endwert ist. Ist das der Fall, wird die Anweisung bzw. Verbundanweisung hinter dem do bearbeitet. Im Anschluß daran wird die Kontrollvariable **um 1 erhöht** und mit dem Endwert auf die Kleiner-Gleich-Bedingung verglichen. Übersteigt der Inhalt der Kontrollvariable den Endwert, so wird die FOR-Schleife verlassen.

Der nachstehende Programmablaufplan verdeutlicht ebenfalls die Ablaufsteuerung bei einer FOR-Anweisung.



Falls es nach der Problemstellung erforderlich ist, daß der Schleifenanfangswert größer ist als der Endwert, so muß die Anweisungsform

for Variable := Anfangswert **downto** Endwert **do** Anweisung

verwendet werden. Bei dieser Form der FOR-Anweisung wird die Kontrollvariable bei jedem Schleifendurchlauf **um 1 vermindert.** Die Schleife wird verlassen, wenn der Endwert unterschritten wird.
Mittels der FOR-Anweisung können Sie Programme elegant gestalten, weil die Erhöhung des Wertes in der Kontrollvariablen und ihr Vergleich mit dem Endwert automatisch erfolgt. Dabei müssen Sie jedoch folgende Randbedingungen beachten:

(1) Der Wert der Kontrollvariablen darf innerhalb des Schleifenkörpers nicht verändert werden. Es ist also nicht statthaft, die Schrittweite zu variieren.
(2) Während der Laufzeit der FOR-Schleife hat eine Änderung des Endwertes keine Auswirkung auf die Schleifendurchläufe. Ihre Anzahl wird nämlich vor dem ersten Durchlauf festgelegt.

Nach Beendigung der FOR-Anweisung ist der Inhalt der Kontrollvariablen undefiniert. Er weist irgendeinen zufälligen Wert auf.

5.3.1 Zusammenfassung

- Für die Anwendung der zählergesteuerten Schleife ist erforderlich, daß die Anzahl der Iterationen vor Beginn der Schleifenbearbeitung bekannt ist.
- Für die zählergesteuerte Schleife gibt es in Pascal zwei Formen:
  (a) **for** Kontrollvariable := Anfangswert **to** Endwert **do** Anweisung
  (b) **for** Kontrollvariable := Anfangswert **downto** Endwert **do** Anweisung
- Die Datentypen von Kontrollvariable, Anfangswert und Endwert müssen übereinstimmen. Sie müssen ordinal sein.

5.3.2 Übungen

1. Aufgabe
Entwickeln Sie ein Programm, mit dem die Summe der Zahlen

1 + 2 + 3 + 4 + ... + 97 + 98 + 99 + 100

gebildet wird.

2. Aufgabe

Warum läßt sich das Programm zu Aufgabe 2 der Übung 5.2.2 nicht unter Anwendung einer FOR-Anweisung schreiben?

5.3.3 Lösungen

1. Aufgabe

```
program Zahlen_Summierung_3;
var Zahl, Summe: integer;
begin
  Summe := 0;
  for Zahl := 1 to 100 do
    Summe := Summe + Zahl;
  writeln ('Die Summe der Zahlen beträgt',Summe:8);
end.
```

2. Aufgabe

Voraussetzung für die Anwendung der FOR-Anweisung ist, daß die Anzahl der Schleifendurchläufe schon vor Beginn der Iteration festliegt. Das ist aber nach der Aufgabenstellung nicht gegeben.

5.4 Sprunganweisung

5.4.1 GOTO

Im Zusammenhang mit Schleifenkonstruktionen kann der Fall auftreten, daß eine weitere Bearbeitung der Schleife nicht mehr sinnvoll ist. Man möchte deshalb auf kürzestem Wege die Schleife verlassen. In einer solchen Situation kann die Sprunganweisung **goto** recht hilfreich sein.
Mittels einer GOTO-Anweisung kann man das Programm veranlassen, aus der Schleife herauszuspringen und an einer anderen Stelle fortzufahren. Der Aufbau der GOTO-Anweisung ist einfach:

goto Marke

Hinter dem Schlüsselwort **goto** ist eine Sprungmarke anzugeben. Marke kann ein Name oder eine ganzzahlige Zahl sein. Die Marke

muß im Vereinbarungsteil mittels des Schlüsselwortes **label** vereinbart werden:

label Marke;

Die Stelle innerhalb des Programms, zu der durch die **goto**-Anweisung verzweigt werden soll, wird durch

Marke:

kenntlich gemacht.
Folgendes Beispiel verdeutlicht die Verwendung einer GOTO-Anweisung in einer Schleifenkonstruktion:

***Programmbeispiel* Abstimmsumme**
In einer Buchhaltung werden bis zu 50 Belege zu einem Abstimmkreis zusammengefaßt. Die Beträge der Belege werden eingegeben und addiert, um Abstimmungssummen zu bilden. Da es aber auch vorkommen kann, daß zu einem Abstimmkreis weniger als 50 Belege gehören, wird nach Eingabe des letzten Betrags eine Null eingegeben, um das Programm vorzeitig beenden zu können.
Das Programm dazu lautet:

```
 0  program Abstimmsumme;
 1  label finito;
 2  var n: integer,
 3      Betrag, Summe: real;
 4  begin
 5  Summe := 0;
 6  for n := 1 to 50 do
 7    begin
 8    write ('Eingabe Betrag, bei Ende 0 eingeben');
 9    readln (Betrag);
10    if Betrag = 0
        then goto finito;
11    Summe := Summe + Betrag;
12    end;
13  finito: writeln('Abstimmsumme:',Summe:10:2)
14  end.
```

Die Sprungmarke lautet in diesem Programm «finito». Sie wurde unter diesem Namen in der Zeile 1 vereinbart. In Zeile 10 ist die Sprunganweisung formuliert mit **goto finito**. Wenn als Endemerkmal für Betrag eine Null eingegeben wurde, verzweigt das Programm zu der Anweisung, die durch die Sprungmarke **finito:** gekennzeichnet ist, und druckt die Abstimmsumme aus.

Falls die Sprunganweisung nicht zur Geltung kommt, weil genau 50 Beträge eingegeben wurden, wird ebenfalls die Abstimmsumme in Zeile 13 ausgedruckt. Daß zu Beginn dieser Zeile eine Marke steht, hat darauf keinen Einfluß. Eine wichtige Nebenbedingung für den Einsatz von GOTO-Anweisungen ist jedoch zu beachten:

> Wenn in einem Programm die Unterprogrammtechnik eingesetzt wird, darf die GOTO-Anweisung nicht dazu genutzt werden, um von einem Unterprogramm ins Hauptprogramm oder in ein anderes Unterprogramm zu springen.
> Eine Sprungmarke, die ja mittels label zu vereinbaren ist, gilt nur in dem Programmblock, in dem sie vereinbart worden ist.

Sprunganweisungen sind möglichst zu **vermeiden**. Sie verleiten leicht zu einer schlecht strukturierten Programmierung.

5.4.2 BREAK

Ab der Version 7 verfügt Turbo-Pascal über eine **BREAK**-Prozedur, die auf REPEAT-, WHILE- und FOR-Anweisungen angewandt werden kann. BREAK beendet eine Schleife. Es wird anschließend mit der Anweisung fortgesetzt, die dem Wiederholungsblock folgt. Unter Anwendung von BREAK lautet das Programm Abstimmsumme:

```
 0  program Abstimmsumme_mit_Break;
 1  var n: integer,
 2      Betrag, Summe: real;
 3  begin
 4  Summe := 0;
 5  for n := 1 to 50 do
 6    begin
 7    write ('Eingabe Betrag, bei Ende 0 eingeben');
 8    readln (Betrag);
 9    if Betrag = 0
        then break;
10    Summe := Summe + Betrag;
11    end; (* Ende der FOR-Anweisung *)
12  writeln('Abstimmsumme:',Summe:10:2)
13  end.
```

Wird in Betrag eine Null eingegeben, veranlaßt BREAK den Sprung aus der Schleife. Das Programm wird anschließend mit der WRITELN-Anweisung fortgesetzt.

5.4.3 Zusammenfassung

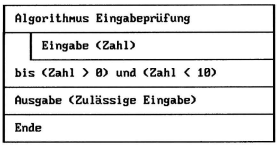
- Die Sprunganweisung lautet:
  goto Marke
  Es wird zu der Anweisung verzweigt, vor der «Marke:» steht.
- Die Sprungmarke ist im Vereinbarungsteil mittels
  label Marke;
  zu vereinbaren.
- BREAK veranlaßt den Sprung aus einer Schleife zu der Anweisung, die der Iterationsanweisung folgt.

5.5 Eingabeschleifen

Die Wiederholungsanweisungen können Sie nun bei verschiedenen Problemstellungen anwenden.

Plausibilitätskontrolle
Bei der Datenverarbeitung im Dialog mit dem Rechner werden die Daten über die Tastatur eingegeben und unmittelbar vom Rechner verarbeitet. Werden aus Versehen Daten fehlerhaft eingegeben, sind natürlich die Ergebnisse falsch, und außerdem kann das zu erheblichen Störungen im Verarbeitungsprozeß führen. Deshalb ist es wichtig, die eingetasteten Daten sofort auf Richtigkeit zu überprüfen. Falls eine Fehleingabe vorliegt, sollte sie möglichst sofort «abgefangen» werden. Am Bildschirm sollte ein Fehlerhinweis erscheinen. Eine erneute Eingabe sollte ermöglicht werden. Diese Art der Datenprüfung nennt man Plausibilitätskontrolle.
Die einfachste Form einer Eingabeschleife wird mittels der REPEAT-Anweisung realisiert.

***Programmbeispiel* Eingabeprüfung**
Es ist eine einstellige ganze Zahl einzugeben, die größer null sein soll. Falls eine falsche Zahl eingetippt wird, ist die Eingabe zu wiederholen. Wurde eine zulässige Zahl eingetastet, erscheint am Bildschirm die Meldung «zulässige Eingabe».
Die Eingabeschleife darf also nur dann verlassen werden, wenn die eingegebene Zahl größer null und kleiner als zehn ist.
Der Algorithmus dazu lautet in der Form des Struktogramms:



Das Programm ist dann wie folgt zu formulieren:

```
0  program Eingabepruefung;
   uses crt;                                   (* ab Version 4 *)
1  var Zahl: integer;
2  begin
3    ClrScr;
4    repeat
5      write ('Eingabe einer Zahl > 0 und < 10 ');
6      readln (Zahl);
7    until (Zahl > 0) and (Zahl < 10);
8    writeln ('Zulässige Eingabe! ');
9  end.
```

Geben Sie das Programm ein, und testen Sie es.

Zusammengesetzter Bedingungsausdruck

Wie in Zeile 7 des obigen Programms Eingabepruefung dargestellt ist, kann ein Bedingungsausdruck auch aus mehreren Teilbedingungen zusammengesetzt werden, die durch AND (deutsch: und) oder OR (deutsch: oder) logisch miteinander verbunden werden. Die Teilbedingungen, aus denen sich der Bedingungsausdruck zusammensetzt, werden in Klammern gesetzt.

Ein Bedingungsausdruck, dessen Teilbedingungen durch AND miteinander verbunden sind, hat nur dann den Wahrheitswert true, wenn alle Teilbedingungen wahr sind. Der Bedingungsausdruck

(Zahl > 0) and (Zahl < 10)

ist also nur dann erfüllt, wenn der Inhalt der Variablen Zahl sowohl größer null als auch kleiner 10 ist. Beinhaltet Zahl den Wert 12, so ist der Bedingungsausdruck nicht erfüllt. Zwar ist der Wert 12 größer als null, er ist aber nicht kleiner als 10.

Wahrheitstafel einer UND-Verknüpfung

Teilbedingung		Gesamtbedingung
A	B	A AND B
w	w	w
w	f	f
f	w	f
f	f	f

Ein Bedingungsausdruck, dessen Teilbedingungen durch OR miteinander verbunden sind, ist schon dann wahr, wenn auch nur eine Teilbedingung erfüllt ist. Oder anders ausgedrückt: er hat nur dann den Wahrheitswert false, wenn keine der Teilbedingungen erfüllt ist.

Beispiel:

```
if (Geschlecht = 'm') OR (Geschlecht = 'M') then write ('männlich');
```

Wahrheitstafel einer ODER-Verknüpfung

Teilbedingung		Gesamtbedingung
A	B	A OR B
w	w	w
w	f	w
f	w	w
f	f	f

Verneinung von Bedingungen:
Schließlich besteht die Möglichkeit, mittels des logischen Operators NOT eine Abfragebedingung umzugestalten.

Beispiel:

	gleichbedeutend mit
if Richtig = false then...	if NOT Richtig then...
if A >= 3 then...	if NOT A < 3 then...
if B <> C then...	if NOT A = C then...

«Stehender» Bildschirm: GOTOXY

Die obige Eingabeschleife ist nicht sehr elegant, zumal sie den ganzen Bildschirm vollschreibt. Besser ist es, wenn die Eingabe stets an derselben Stelle auf dem Bildschirm erscheint. Das erreicht man durch den Cursorbefehl

gotoxy (Spalte,Zeile)

Spalte und Zeile sind Integer-Ausdrücke. Sie geben an, wohin der Cursor gestellt werden soll.

Beispiel:

```
gotoxy (3,8)
```

Der Cursor wird in die 3. Spalte und 8. Zeile gesetzt. Soll der Cursor in die linke obere Ecke gesetzt werden, lautet der Befehl gotoxy(1,1). In die rechte untere Ecke setzt man ihn mit gotoxy(80,25).
Fügen Sie in das Programm Eingabepruefung den Cursorsteuerbefehl hinter das repeat ein:

```
4 repeat
    gotoxy (3,8);
5 write ('Eingabe einer Zahl > 0 und < 10 ');
```

Wenn Sie nun das Programm starten, erfolgt die Ausgabe stets auf derselben Stelle am Bildschirm.
Um einen alten, fehlerhaften Eingabewert zu löschen, verwenden Sie den Befehl

ClrEol

(clear end of line; deutsch: Löschen bis zum Zeilenende).
Mit ClrEol wird ab der Cursorstelle die Zeile nach rechts am Bildschirm gelöscht, ohne daß der Cursor dabei verstellt wird. Der Befehl ClrEol muß unmittelbar vor der Anweisung readln(Zahl) stehen.
Mit diesen beiden Änderungen lautet das Programm für die Eingabeschleife:

```
0  program Eingabepruefung;
   uses crt;                                    (* ab Version 4 *)
1  var Zahl: integer;
2  begin
3    ClrScr;
4    repeat
5      gotoxy (3,8);
6      write ('Eingabe einer Zahl > 0 und < 10 ');
7      ClrEol;
8      readln (Zahl);
9    until (Zahl > 0) and (Zahl < 10);
10   writeln ('Zulässige Eingabe! ');
11 end.
```

Damit der Benutzer des Programms auf eine fehlerhafte Eingabe aufmerksam gemacht wird, fehlt noch die Ausgabe eines Fehlerhinweises. Sie ließe sich mit Hilfe einer IF-Anweisung in das obige Programm hineinprogrammieren. Die Behandlung einer fehlerhaften Eingabe läßt sich aber leichter programmieren, wenn statt der REPEAT-Anweisung eine WHILE-Anweisung verwendet wird, wie das nachstehende Programm zeigt.

```
0  program Eingabepruefung2;
   uses crt;                                    (* ab Version 4 *)
1  var Zahl: integer;
2  begin
3    ClrScr;
4    gotoxy (3,8);
5    write ('Eingabe einer Zahl > 0 und < 10 ');
6    ClrEol;
7    readln (Zahl);
8    while (Zahl < 1) or (Zahl > 9) do
9      begin
10       write (' falsch!'); delay (300);
11       gotoxy (35,8); ClrEol;
12       readln (Zahl);
13     end;
14   writeln;
15   writeln ('Zulässige Eingabe! ');
16 end.
```

Erläuterungen zum Programm:
Dieses Programm baut auf der WHILE-Anweisung auf. Es ist etwas länger als das vorherige und sieht vielleicht auch komplizierter aus, ist es aber nicht.
Nach der Vorbereitung des Bildschirms in den Zeilen 3 bis 6 erfolgt in Zeile 7 die Eingabe der Zahl. Ist die Zahl größer als 0 und kleiner als 10, wird mit der Anweisung in Zeile 14 fortgefahren.

Bei einer zulässigen Eingabe wird die WHILE-Anweisung übergangen. Die Anweisungen in der WHILE-Schleife werden nur dann bearbeitet, wenn der eingegebene Wert nicht zulässig ist. Nicht zulässig ist der Wert, wenn er kleiner als 1 oder größer als 9 ist.

Wurde beispielsweise in Zeile 7 der Wert 23 eingegeben, so ist der Wert zwar nicht kleiner als 1, aber er ist größer als 9. Die zweite Teilbedingung des Bedingungsausdrucks ist also erfüllt. Somit ist die Schleifeneintrittsbedingung erfüllt, da die beiden Teilbedingungen durch or verknüpft sind. Zur Erinnerung: Bei einer Oder-Verknüpfung von Bedingungen muß nur eine Teilbedingung erfüllt sein, damit die gesamte Oder-Bedingung den Wahrheitswert true hat.

Als erste Aktion innerhalb der WHILE-Schleife wird der Hinweis «falsch!» hinter die Eingabe geschrieben. Damit der Benutzer den Hinweis auch bemerkt, wird das Programm 300 Millisekunden (= 0,3 Sekunden) angehalten. Der Befehl für dieses Anhalten des Programms lautet

```
delay (i)
```

i ist ein ganzzahliger Ausdruck und gibt an, für wieviel Millisekunden das Programm eine Pause machen soll. Nach Ablauf der Zeitspanne fährt es mit der nächsten Anweisung fort.

In dem Cursorsteuerbefehl

```
gotoxy (35,8)
```

in Zeile 11 wird der Cursor auf die Eingabestelle zurückgesetzt; dann wird der rechts vom Cursor stehende Text durch die Anweisung ClrEol gelöscht.

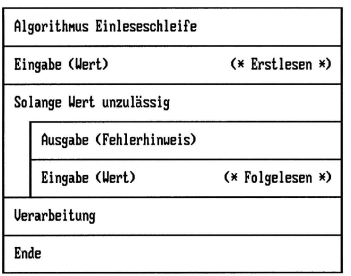
Mit der Lese-Anweisung in Zeile 12 wird die erneute Eingabe ermöglicht. Die nächste Anweisung, die nach der Eingabe ausgeführt wird, ist die WHILE-Anweisung in Zeile 8. Unmittelbar nach der Eingabe des Werts wird er in der Schleifeneintrittsbedingung überprüft. Ist der Wert wieder nicht zulässig, wird die Schleife erneut durchlaufen, und es kann in Zeile 12 wiederum ein Wert eingegeben werden. Erst wenn es sich um einen zulässigen Wert handelt, wird die WHILE-Schleife verlassen, und das Programm wird mit der Anweisung in Zeile 14 fortgesetzt.

Erstlesen – Folgelesen

In diesem Eingabeprogramm wurde an zwei Stellen die Anweisung

```
readln (Zahl)
```

gegeben, nämlich in Zeile 7 unmittelbar vor der Schleife und in Zeile 12 am Ende der Schleife. Dadurch ist gewährleistet, daß jede Eingabe sofort anschließend auf Zulässigkeit geprüft wird.
Die Lese-Anweisung in Zeile 7 wird **Erstlesen** genannt, die in Zeile 12 **Folgelesen**. Die Anweisung in Zeile 7 wird nur einmal ausgeführt. War die Eingabe dort fehlerhaft, werden alle weiteren Eingaben über die READLN-Anweisung in Zeile 12 abgewickelt. Diese Methode des **Erstlesens** und **Folgelesens** wird sehr häufig zur Gestaltung von Eingabeschleifen angewendet. Deshalb wird sie nachstehend in verallgemeinerter Form graphisch dargestellt.
Darstellung der Eingabeschleife als Struktogramm:



***Programmbeispiel* Konzentration**

Mit den oben neu behandelten Befehlen zur Bildschirmsteuerung und dem Erstlesen-Folgelesen-Algorithmus wird ein kleines Konzentrationsspiel programmiert.
Spielregeln: Auf dem Bildschirm erscheint an einer zufälligen Position eine Zahl zwischen 0 und 9. Der Spieler soll die Zahl erkennen und eingeben. Gibt er eine falsche Zahl ein, erscheint die Meldung «falsch», und er kann die Eingabe wiederholen. Insgesamt werden ihm drei weitere Versuche zugestanden.

Problemanalyse

(1) Bildung einer Zufallszahl

Im Sprachumfang von Turbo-Pascal ist eine Anweisung vorhanden, mit der durch einen Zufallszahlengenerator eine ganzzahlige Zufallszahl gebildet wird. Soll die Zahl zwischen 0 und 9 liegen, so lautet der Befehl:

```
random (10)
```

Mit der Anweisung

```
Zufallszahl := random (10)
```

kann sich in der ganzzahligen Variablen ein Wert größer als oder gleich 0 und kleiner als 10 bilden. Der Zufallszahlengenerator wird vorher durch **randomize** aktiviert.

(2) Bildschirmpositionierung

Die Zufallszahl soll irgendwo auf dem Bildschirm erscheinen. Den Befehl gotoxy(Spalte,Zeile) zur Cursorsteuerung kennen Sie bereits. Der Bildschirm hat 25 Zeilen und 80 Spalten. Zur Festlegung der Stelle, an der die Zahl ausgegeben werden soll, können Sie sich die Random-Funktion zunutze machen.

```
Zeile := random (24) + 1
Spalte := random (79) + 1
```

Bei dieser Zeilen- und Spaltenermittlung wird berücksichtigt, daß die Zeilen- und Spaltennumerierung jeweils mit 1 beginnt. Die linke obere Ecke des Bildschirms hat die Koordinaten (1,1). Außerdem ist sichergestellt, daß der Ausgabepunkt nicht außerhalb des Bildschirms liegen wird.

(3) Zeitspanne

Die Zufallszahl darf nur für einen kurzen Augenblick gezeigt werden. Dann muß sie wieder gelöscht werden. Am einfachsten programmiert man das auf folgende Weise: Nach Ausgabe der Zufallszahl wird das Programm angehalten. Der Befehl, besser: die Standardprozedur, lautet **delay**. Mit **delay(280)** wird die Programmbearbeitung für 280 Millisekunden unterbrochen. Anschließend wird mit ClrScr der Bildschirm gelöscht.

Datenanalyse

Ausgabedaten: Auszugeben sind die Textkonstanten

```
'richtig'
'falsch'
```

oder

```
'nicht erraten'
```

Eingabedaten: Es ist eine Zahl einzugeben. Sie wird in einer Integer-Variablen gespeichert.

```
EingabeZahl: integer.
```

Beziehungen zwischen den Eingabe- und Ausgabedaten: Wenn EingabeZahl = Zufallszahl, wird der Text 'richtig' ausgegeben. Stimmen die beiden Zahlen nicht überein, wird der Text 'falsch' angezeigt. Wenn nach drei weiteren Versuchen nicht die richtige Zahl eingegeben wurde, erscheint auf dem Bildschirm der Text 'nicht erraten'. Um die Anzahl der Versuche nachhalten zu können, muß noch eine weitere Variable «Fehlversuche» mitgeführt werden.

Algorithmus

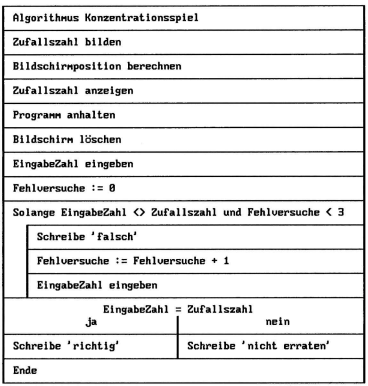
Pseudocode:

```
Algorithmus Konzentrationsspiel
Beginn
  Zufallszahl bilden
  Bildschirmposition berechnen
  Zufallszahl anzeigen
  Programm anhalten
  Bildschirm löschen
  Zahl eingeben
  Anzahl der Fehlversuche auf 0 setzen
  Solange (EingabeZahl <> Zufallszahl) und (Fehlversuche < 3)
     tue
        Text 'falsch' schreiben
             Fehlversuche um 1 erhöhen
     Zahl eingeben
  Wenn EingabeZahl = Zufallszahl
```

dann schreibe Text 'richtig'
sonst schreibe Text 'nicht erraten'

Ende

Als Struktogramm hat der Algorithmus folgende Form:



- Programm

```
program Konzentration;
uses crt;                                              (* ab Version 4 *)
const Verzoegerung = 200;
var Zufallszahl, EingabeZahl: integer;
    Fehlversuche, Zeile,Spalte: integer;
begin
  ClrScr;
  zeile  := random (24); Zeile  := Zeile  + 1;
  Spalte := random (79); Spalte := Spalte + 1;
  Zufallszahl := random (10);
  gotoxy (Spalte,Zeile);
  write (Zufallszahl); delay (Verzoegerung); ClrScr;
  gotoxy (12,10); write ('Eingabe der Zahl '); readln (EingabeZahl);
  Fehlversuche := 0;
  while (EingabeZahl <> Zufallszahl) and (Fehlversuche < 3) do
    begin
      gotoxy (33,10); write ('falsch!'); delay(100);
      Fehlversuche := Fehlversuche + 1;
      gotoxy (12,10); write ('Eingabe der Zahl '); ClrEol;
      readln (EingabeZahl);
    end;
  if EingabeZahl = Zufallszahl
    then write (' richtig!')
    else write (' nicht erraten! ');
end.
```

Dieses Spiel kann man noch variieren. So könnten Sie die Zeit für die Verzögerung variabel gestalten. Dazu müßte die Konstantenvereinbarung gelöscht und eine Eingabeanweisung

readln (Verzoegerung)

eingebaut werden.

5.5.1 Zusammenfassung

- Eingabedaten sollten vor ihrer weiteren Verarbeitung auf Plausibilität geprüft werden.
- Mittels **and** und **or** können Bedingungsausdrücke verknüpft werden.
- Die Methode «Erstlesen – Folgelesen» ermöglicht die unmittelbare Prüfung von Eingabedaten.
- **gotoxy (Spalte, Zeile)** stellt den Cursor auf eine bestimmte Bildschirmposition. Spalte und Zeile sind Integer-Ausdrücke.
- **ClrEol** löscht eine Bildschirmzeile ab der Cursorposition bis zum Zeilenende.

- **delay (i)** hält das Programm für i Millisekunden an. i ist ein Integer-Ausdruck.
- **random (i)** erzeugt eine Zufallszahl, die zwischen 0 und i liegt. i ist ein Integer-Ausdruck.

5.5.2 Übungen

1. Aufgabe

In der Buchhaltung eines Unternehmens wird ein Programm benötigt, mit dem die Anschaffungsbeträge der sogenannten geringwertigen Wirtschaftsgüter eingegeben und summiert werden. Ein Wirtschaftsgut gilt als geringwertig, wenn der Anschaffungspreis nicht höher als 800 DM ist.
Die Eingabe und Aufsummierung der Beträge soll so lange vorgenommen werden, bis als Betrag eine Null eingegeben wurde.
Falls ein Betrag eingegeben wurde, der kleiner als null oder größer gleich 800 ist, soll am Bildschirm ein Fehlerhinweis für 0,5 Sekunden erscheinen. Der Betrag darf nicht in die Summierung einbezogen werden.
Zu dieser Problemstellung liegt folgendes Programm vor. Ergänzen Sie das Programm um die fehlenden Bedingungsausdrücke, geben Sie es in den Rechner ein, und testen Sie es.

```
program Aufsummierung;
var Betrag, Summe: real;
begin
  ClrScr;
  Summe := 0;
  gotoxy(5,3);
  write ('Aufsummierung beliebig vieler Zahlen zwischen 0 und
  800');
  gotoxy(10,5); write ('Betrag');
  gotoxy(10,6); write ('(Ende = 0)');
  gotoxy(10,8); write ('Summe:');
  gotoxy(18,5); ClrEol;
  readln (Betrag);
  while _______ do
    begin
    if (_____) or (_______)
```

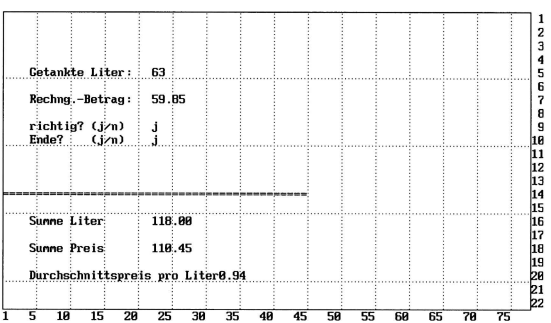
```
            then begin
                gotoxy(30,5); write ('Unzulaessige Eingabe!');
                delay (500);
                end
            else begin
                Summe := Summe + Betrag;
                gotoxy(16,8); write (Summe:10:2);
                end;
        gotoxy(18,5); ClrEol;
        readln (Betrag);
    end;
    gotoxy (10,12); write ('Ende des Programms');
end.
```

2. Aufgabe

Es ist ein Programm zu entwickeln, mit dem der durchschnittliche Kraftstoffpreis pro Liter (gewogener Durchschnittspreis) ermittelt wird. Dazu wird eingegeben:

Getankte Liter
Rechnungsbetrag

Bildschirmaufbau:



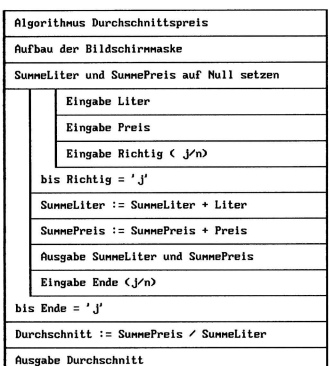
Nach der Eingabe der beiden obigen Werte ist die Richtigkeit der Eingabe zu bestätigen. Sind die Eingaben nicht korrekt, werden sie wiederholt.

Falls die Eingaben richtig sind, sind die Liter- und Betragsangaben zu summieren. Die Summen sind auf dem Bildschirm anzuzeigen (siehe Bildschirmmaske). Weitere Werte können eingegeben werden, wenn auf die Endeabfrage mit einem «n» geantwortet wird.

Wird bei der Endeabfrage ein «j» eingetastet, so wird der Durchschnittspreis angezeigt. Der Durchschnittspreis errechnet sich aus

$$\text{Durchschnittspreis} = \frac{\text{Summe der Preise}}{\text{Summe der Liter}}$$

Folgendes Struktogramm skizziert den Algorithmus:



5.5.3 Lösungen

1. Aufgabe

```
program Aufsummierung;
uses crt;                                              (* ab Version 4 *)
var Betrag, Summe: real;
begin
  ClrScr;
  Summe := 0;
  gotoxy(5,3);
  write ('Aufsummierung beliebig vieler positiver Zahlen
         zwischen 0 und 800');
  gotoxy(10,5); write ('Betrag');
  gotoxy(10,6); write ('(Ende = 0)');
  gotoxy(10,8); write ('Summe:');
  gotoxy(18,5); ClrEol;
  readln (Betrag);
  while Betrag <> 0 do
    begin
      if (Betrag < 0) or (Betrag >= 800)
        then begin
               gotoxy(30,5); write ('Unzulaessige Eingabe!');
               delay (500);
             end
        else begin
               Summe := Summe + Betrag;
               gotoxy(16,8); write (Summe:10:2);
             end;
      gotoxy(18,5); ClrEol;
      readln (Betrag);
    end;
  gotoxy (10,12); write ('Ende des Programms');
end.
```

2. Aufgabe

```
program Durchschnittlicher_Kraftstoffpreis;
uses crt;                                          (* ab Version 4 *)
var Liter, SummeLiter, Preis, SummePreis, Durchschnitt: real;
    Richtig, Ende: char;
begin
  ClrScr;
  gotoxy (5, 5); write ('Getankte Liter:');
  gotoxy (5, 7); write ('Rechng.-Betrag:');
  gotoxy (5, 9); write ('richtig? (j/n):');
  gotoxy (5,10); write ('Ende (j/n)    :');
  gotoxy (1,14);
  write ('=============================================');
  gotoxy (5,16); write ('Summe Liter:');
  gotoxy (5,18); write ('Summe Preis:');
  gotoxy (5,20); write ('Durchschnittspreis pro Liter:');
  SummeLiter := 0; SummePreis := 0;
  repeat
    repeat
      gotoxy (23,5); ClrEol; readln (Liter);
      gotoxy (23,7); ClrEol; readln (Preis);
      gotoxy (23,9); ClrEol; readln (Richtig);
    until Richtig = 'j';
    SummeLiter := SummeLiter + Liter;
    SummePreis := SummePreis + Preis;
    gotoxy (23,16); write (SummeLiter:8:2);
    gotoxy (23,18); write (SummePreis:8:2);
    gotoxy (23,10); readln (Ende);
  until Ende = 'j';
  Durchschnitt := SummePreis / SummeLiter;
  gotoxy (33,20); write (Durchschnitt:8:2);
end.
```

5.6 Erstellen von Tabellen

Eine Tabelle besteht in der Regel aus einem Tabellenkopf und einem Tabellenrumpf. Der Tabellenkopf beinhaltet die Überschriften zu den Spalten des Tabellenrumpfes. Im Tabellenrumpf werden die einzelnen Werte spaltenweise ausgegeben.

Tabellenaufbau			
Tabellenkopf	aaaaaa	bbbbbb	ccccccc
	12345	12345	12345
Tabellenrumpf	12345	12345	12345
	1234567	1234567	1234567

Die Programmierung einer tabellenförmigen Ausgabe wird am Beispiel einer Abschreibungstabelle aufgezeigt.

***Programmbeispiel* Abschreibung**

Wenn ein Unternehmen ein Auto kauft, so verliert dieses im Laufe der Jahre an Wert. Nach beispielsweise 5 Jahren ist es «abgeschrieben». Die Wertverluste werden in Abschreibungstabellen erfaßt.

Nach dem deutschen Steuerrecht gibt es verschiedene Methoden, die Abschreibung zu berechnen. Hier soll die lineare Abschreibungsmethode angewendet werden. Wie eine Abschreibungstabelle nach der linearen Abschreibungsmethode aufgebaut wird, wird im folgenden dargelegt.

Es wird angenommen, daß der Anschaffungswert des Autos 25000 DM beträgt. Es wird erwartet, daß das Auto 5 Jahre genutzt werden kann. Dann beträgt die jährliche Abschreibung:

$$\text{jährliche Abnutzung} = \frac{\text{Anschaffungswert}}{\text{Nutzungsdauer}} = \frac{25000}{5} = 5000 \text{ DM}$$

Die Abschreibungstabelle sieht dann wie folgt aus:

Abschreibungstabelle			
Jahre	**Anfangswert**	**Abschreibung**	**Endwert**
1	25000.00	5000.00	20000.00
2	20000.00	5000.00	15000.00
3	15000.00	5000.00	10000.00
4	10000.00	5000.00	5000.00
5	5000.00	5000.00	0.00

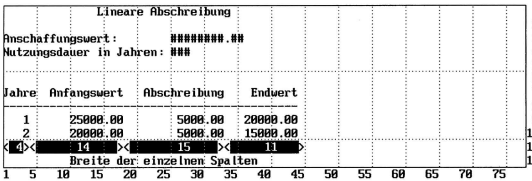
Problemstellung

Es ist eine Abschreibungstabelle zu erstellen. Von dem Abschreibungsgegenstand sind der Anschaffungswert und die erwartete Nutzungsdauer bekannt.

Problemanalyse

Für jedes Jahr der Nutzungszeit ist eine Tabellenzeile auszugeben. Sie enthält Angaben zum Nutzungsjahr, Jahresanfangswert, Abschreibungshöhe und zum Jahresendwert. Im ersten Jahr ist der Jahresanfangswert gleich dem Anschaffungswert. In allen weiteren Jahren ist er gleich dem Endwert des Vorjahres.

Tabellenformat



Datenanalyse

Ausgabedaten: Tabellenkopf:
Jahre Anfangswert Abschreibung Endwert :
Textkonstante
Tabellenrumpf:
Jahre: integer
Anfangswert: real
Abschreibung: real
Endwert: real

Eingabedaten: Anschaffungswert: real
Nutzungsdauer: integer

Beziehungen zwischen den Eingabe- und Ausgabedaten:

Anfangswert := Anschaffungswert (im 1. Jahr)
Anfangswert := Endwert (in den Folgejahren)
Abschreibung := Anschaffungswert / Nutzungsdauer
Endwert := Anfangswert – Abschreibung
Jahre := Jahre + 1

Algorithmus

Zuerst sind die bekannten Daten wie Anschaffungswert und Nutzungsdauer einzugeben. Aus diesen beiden Angaben wird die Abschreibung errechnet. Anschließend kann die Tabelle gedruckt werden.

Die Berechnung der Werte einer Tabellenzeile und ihr Ausdruck wird in einer Schleife vorgenommen. Da mit der Angabe zur Nutzungsdauer bekannt ist, wie oft die Schleife durchlaufen werden

muß, kann hier die FOR-Anweisung verwendet werden. Deshalb kann die Anweisung Jahre := Jahre + 1 entfallen.
Der Algorithmus (Lösungsweg) lautet dann

```
Pseudocode:
Algorithmus Lineare_Abschreibung
Beginn
Eingabe Anschaffungswert
Eingabe Nutzungsdauer
Berechne Abschreibung
Anfangswert erhält den Wert von Anschaffungswert
Für Jahre := 1 bis Nutzungsdauer
   tue
   Endwert := Anfangswert – Abschreibung
   Drucke Tabellenzeile
   Anfangswert := Endwert
Ende
```

Struktogramm

Programm

```
0  program Lineare_Abschreibung;
   uses crt;                                          (* ab Version 4 *)
1  var Jahre, Nutzungsdauer: integer;
2      Anschaffungswert, Anfangswert, Abschreibung, Endwert: real;
3  begin
4    ClrScr;
5    writeln ('              Lineare Abschreibung');
6    writeln;
7    write ('Anschaffungswert:        '); readln (Anschaffungswert);
8    write ('Nutzungsdauer in Jahren: '); readln (Nutzungsdauer);
9    Abschreibung := Anschaffungswert / Nutzungsdauer;
10   writeln;writeln;
11   Anfangswert := Anschaffungswert;
12   writeln ('Jahre   Anfangswert   Abschreibung   Endwert');
13   writeln ('-------------------------------------------');
14   for Jahre := 1 to Nutzungsdauer do
15     begin
16       Endwert := Anfangswert - Abschreibung;
17       writeln (Jahre:4, Anfangswert:14:2, Abschreibung:15:2,
                  Endwert:11:2);
18       Anfangswert := Endwert;
19     end;
20 end.
```

Geben Sie das Programm ein, und testen Sie es!
Anmerkungen:
(1) In der Ausgabeanweisung (Zeile 17)

```
writeln (Jahre:4, Anfangswert:14:2, Abschreibung:15:2, End-
        wert:11:2);
```

sind für die einzelnen Variablen sehr große Druckmaskenformate angegeben worden. Dies geschah nicht deshalb, weil so große Zahlenwerte erwartet werden. Vielmehr können Sie in Turbo-Pascal über die Druckmaskenformate eine spaltengerechte Ausgabe der Tabellenzeile erreichen.
(2) Die Anweisung (Zeile 18)

```
Anfangswert := Endwert
```

dient dazu, den Endwert eines Nutzungsjahres auf den Anfangswert des nächsten Nutzungsjahres zu übertragen. Diesen Vorgang nennt man **Wertfortschreibung**.

5.6.1 Zusammenfassung

- Eine Tabelle besteht aus Tabellenkopf und Tabellenrumpf. Die Ausgabe des Tabellenrumpfs erfolgt über eine Wiederholungsanweisung.
- Die spaltengerechte Ausgabe einer Tabellenzeile wird durch entsprechende Druckmaskenangaben erreicht.
- Sind in einem Programm Wertfortschreibungen vorzunehmen, so muß am Ende der Wiederholungsanweisung der Endwert auf den Anfangswert zugewiesen werden.

5.6.2 Übungen

1. Aufgabe

Bei Kreditverhandlungen setzt ein Kreditinstitut ein Programm ein, das eine Tilgungstabelle erzeugt. An Hand der Tilgungstabelle kann der Kunde ersehen, wie hoch die jährlichen Rückzahlungsverpflichtungen sind.

Eingabe: Darlehensbetrag 100000
Tilgungsjahre 10

Jahr	Anfangswert	Zinsen	Tilgung	Endwert	Zahlbetrag
1	100000.00	8000.00	10000.00	90000.00	18000.00
2	90000.00	7200.00	10000.00	80000.00	17200.00
3	80000.00	6400.00	10000.00	70000.00	16400.00
4	70000.00	5600.00	10000.00	60000.00	15600.00
5	60000.00	4800.00	10000.00	50000.00	14800.00
6	50000.00	4000.00	10000.00	40000.00	14000.00
7	40000.00	3200.00	10000.00	30000.00	13200.00
8	30000.00	2400.00	10000.00	20000.00	12400.00
9	20000.00	1600.00	10000.00	10000.00	11600.00
10	10000.00	800.00	10000.00	0.00	10800.00

Einzugeben sind:
Kreditbetrag
Anzahl der Jahre, in denen der Kunde den Kredit zurückzahlen will

Der Betrag, mit dem der Kredit getilgt wird, ist wie folgt zu berechnen:

$$\text{Tilgung} = \frac{\text{Kredit}}{\text{Tilgungsjahre}}$$

Der jährliche Zahlungsbetrag ergibt sich aus Tilgung zuzüglich Zinsen. Der Zinssatz beträgt 8 %.

Vervollständigen Sie das nachstehende Programm:

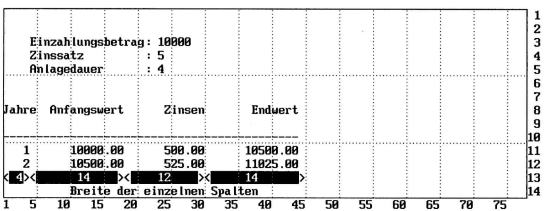
```
program Darlehen;
const Zinssatz = 8;
var Jahre, Zeit : integer;
  Zahlbetrag, Kredit, Anfangswert, Endwert, Zinsen, Tilgung
real;
begin
  ClrScr;
  write ('Eingabe Darlehensbetrag:'); readln (Kredit);
  write ('Eingabe Tilgungsratetrag :'); readln (Jahre);
  Tilgung := Kredit / Jahre;
  Anfangswert := _____;
  writeln  ('Jahr Anfangswert Zinsen Tilgung Endwert
          Zahlungsbetrag');
  writeln ('_____');
  for Zeit := 1 to _____ do
  begin
    Zinsen := Anfangswert * Zinssatz / 100;
    Zahlbetrag := _____;
    Endwert := Anfangswert – Tilgung;
    write (Zeit:4, Anfangswert:10:2, Zinsen:10:2, Tilgung:10:2);
    writeln (Endwert:10:2, Zahlbetrag:10:2);
    _____ := _____;
    end;
end.
```

2. Aufgabe

Es ist ein Programm zu entwickeln, mit dem das Anwachsen des Kontostandes auf einem Sparkonto durch Zinszuschreibung nachvollzogen wird.

Einzugeben sind: Einmaliger Sparbetrag
Zinssatz
Spardauer

Ausgabe (Zahlenbeispiel):



a) Führen Sie die Datenanalyse durch.
b) Entwickeln Sie den Algorithmus.
c) Erstellen Sie das Programm, und testen Sie es.

5.6.3 Lösungen

1. Aufgabe

```
program Darlehen;
uses  crt;                                          (* ab Version 4 *)
const Zinssatz = 8;
var Jahre,Zeit : integer;
    Zahlbetrag, Kredit, Anfangswert, Endwert, Zinsen, Tilgung: real;
begin
  ClrScr;
  write ('Eingabe Darlehensbetrag:'); readln (Kredit);
  write ('       Tilgungsjahre  :'); readln (Jahre);
  Tilgung := Kredit / Jahre;
  Anfangswert := Kredit;
  writeln ('Jahr Anfangswert  Zinsen  Tilgung   Endwert  Zahlbetrag');
  writeln ('-------------------------------------------------------');
  for Zeit := 1 to Jahre do
    begin
      Zinsen := Anfangswert * Zinssatz / 100;
      Zahlbetrag := Tilgung + Zinsen;
      Endwert := Anfangswert  - Tilgung;
      write (Zeit:4,Anfangswert:10:2, Zinsen:10:2, Tilgung:10:2);
      writeln (Endwert:10:2, Zahlbetrag:10:2);
      Anfangswert := Endwert;
    end;
end.
```

2. Aufgabe
Struktogramm

Algorithmus Sparen
Eingabe Sparbetrag
Eingabe Zinssatz
Eingabe Spardauer
Anfangswert := Sparbetrag
Ausgabe Tabellenkopf
Für Jahre := 1 bis Spardauer
Zinsen := Anfangswert * Zinssatz / 100
Endwert := Anfangswert + Zinsen
Drucke Tabellenzeile
Anfangswert := Endwert
Ende

Programm

```
program Sparen;
uses crt;                                                    (* ab Version 4 *)
var Zinssatz, Sparbetrag, Anfangswert, Zinsen, Endwert: real;
    Jahre, Spardauer: integer;
begin
  ClrScr;
  gotoxy(5,3); write ('Einzahlungsbetrag: '); readln (Sparbetrag);
  gotoxy(5,4); write ('Zinssatz         : '); readln (Zinssatz);
  gotoxy(5,5); write ('Anlagedauer      : '); readln (Spardauer);
  writeln;writeln;
  Anfangswert := Sparbetrag;
  writeln ('Jahre   Anfangswert     Zinsen       Endwert');
  writeln ('-----------------------------------------------');
  for Jahre := 1 to Spardauer do
    begin
      Zinsen := Anfangswert * Zinssatz / 100;
      Endwert := Anfangswert + Zinsen;
      writeln (Jahre:4,Anfangswert:14:2,Zinsen:12:2,Endwert:14:2);
      Anfangswert := Endwert;
    end;
end.
```

6 ZEICHENKETTE (STRING)

Klein-Gregor ist in der Schule beim Abschreiben erwischt worden. Zur Strafe soll er fünfzigmal den Satz schreiben: «Ich darf nicht abschreiben!» Eine Aufgabe, die einige Zeit in Anspruch nimmt. Wie schnell das mit Hilfe eines Programms geht, können Sie selber ausprobieren.

```
program Strafarbeit;
uses crt;                                        (* ab Version 4 *)
var Zaehler, Anzahl: integer;
    Textzeile: string [80];
begin
  ClrScr;
  writeln ('Wie lautet die Textzeile?');
  readln (Textzeile);
  write ('Wie oft soll die Textzeile geschrieben werden? ');
  readln (Anzahl);
  for Zaehler := 1 to Anzahl do
    writeln (Textzeile);
end.
```

Mit diesem Programm können Sie jeden beliebigen Text, der 80 Zeichen oder weniger umfaßt, in einer gewünschten Anzahl auf den Bildschirm schreiben lassen.

Ausgabe über Drucker
Falls der Text über einen angeschlossenen Drucker ausgegeben werden soll, müssen Sie die Schreibanweisung in der FOR-Schleife wie folgt formulieren:

```
writeln (lst,Textzeile);
```

Jede Ausgabe über den Drucker ist nach folgendem Muster zu programmieren:

write (**lst**, ...)
beziehungsweise
writeln (lst, ...)

lst steht für **lister** (deutsch: Drucker). Mittels lst wird der Rechner angewiesen, die Ausgabe über den Drucker vorzunehmen.
Ab der Version 4 müssen Sie bei Ausgaben über den Drucker hinter uses auch **PRINTER** vereinbaren, also uses CRT, **PRINTER**.

6.1 Datentyp STRING

Der Text, den Sie in dem obigen Programm in die Variable Textzeile eingegeben haben, besteht aus einer Aneinanderreihung von Zeichen. Man bezeichnet sie auch als Zeichenkette. Die Zeichenkette kann aus

numerischen Zeichen wie 1 2 3 4 5 6 7 8 9 0
alphabetischen Zeichen wie A B C... Y Z a b c... y z
Sonderzeichen wie ! § % + * ...

bestehen.
Um Zeichenketten im Computer speichern zu können, bedarf es eines besonderen Datentyps. Bislang haben Sie die Ein-Zeichen-Variable kennengelernt. Dieser Datentyp wird mit **char** bezeichnet. Er belegt im Arbeitsspeicher 1 Byte. In einer Variable vom Typ char kann man also ein Zeichen speichern.

Eine solche Variable vom Datentyp char reicht aber nicht aus, um einen Text, der aus mehreren Zeichen besteht, aufzunehmen. Dazu müßten mehrere Char-Variablen nebeneinander geschaltet werden:

Da zusammengefügte Char-Variablen bei der Programmierung ein bißchen umständlich zu handhaben sind, enthält Turbo-Pascal den Datentyp **string** (deutsch: Kette). Bei der Vereinbarung einer Zeichenketten-Variablen haben Sie zwei Möglichkeiten. Durch

```
var TextZeile: string;
```

vereinbaren Sie eine String-Variable, die 255 Stellen bzw. Bytes umfaßt. Diese Deklarationsweise ist zwar einfach, belegt aber in den meisten Fällen mehr Speicherplatz als erforderlich. Durch die Vereinbarung

```
var TextZeile: string [80];
```

wird der Speicherplatzbedarf für die Variable TextZeile auf 80 Bytes begrenzt. Die Angabe der wirklich benötigten Länge spart also Speicherplatz ein.

Die Längenangabe besteht aus einem ganzzahligen Wert. Sie wird in eckige Klammern hinter das reservierte Wort string gesetzt. Falls auf der Tastatur des Rechners keine eckigen Klammern verfügbar sind, kann man die Längenangabe auch in runde Klammern mit Punkten schreiben.

Beispiel:

```
var Textzeile: string (.80.);
```

Die Länge einer String-Variablen ist frei vereinbar. Sie kann zwischen 1 und 255 Zeichen liegen.

```
var   KleinsteZeichenkette: string[1];
      GroessteZeichenkette: string[255];
```

Umfaßt eine Zeichenkette mehr als 255 Zeichen, so muß sie auf mehrere String-Variablen aufgeteilt werden.
Eine Zeichenkette darf, wie oben bereits erwähnt, auch aus Ziffern bestehen. Es handelt sich dann um eine Ziffern-Zeichenfolge. Dabei ist zu beachten, daß eine Zeichenkette, die beispielsweise aus den Ziffern

1	4	5

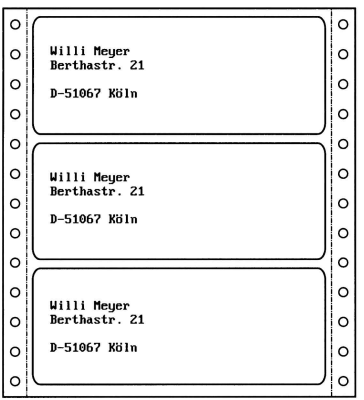
besteht, nicht gleich einer Integer-Variablen mit dem Inhalt

145

ist. Eine Zahl besteht nämlich nicht nur aus einer Kette von Ziffern. Vielmehr besitzt jede Ziffer einer Zahl einen bestimmten Stellenwert. Deshalb kann man eine Zeichenkette, die aus Ziffern besteht, nicht in Rechenoperationen verwenden.

6.2 Verarbeitung von Zeichenketten

Ihre Kenntnisse über Zeichenketten können Sie nun in einem Programm, mit dem Adreßaufkleber erstellt werden, anwenden. Adreßaufkleber haben sich bei vielen Gelegenheiten als nützlich erwiesen. Sie können eine Menge Schreibarbeit ersparen. Viele Computeranwender haben sich bereits einen Stapel Aufkleber erstellt, um sie bei entsprechender Gelegenheit auf Briefbögen, Briefumschläge, Buchdeckel, Koffer usw. zu kleben.



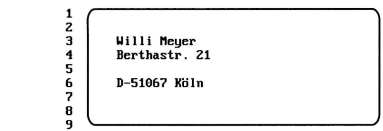
Programmbeispiel Adreßaufkleber

Problemstellung

Dieses Programm soll folgendes leisten: Zunächst gibt der Benutzer den Namen, die Straße sowie Postleitzahl und Wohnort in den Rechner ein. Dann teilt er dem System mit, wie viele Ausdrucke vorgenommen werden sollen. Hat er beispielsweise den Wert 50 eingegeben, so sollen anschließend 50 Aufkleber beschrieben werden.
Die Länge der einzelnen Daten sei wie folgt festgelegt:

Daten	max. Zeichlänge
Name	40
Straße/Hausnummer	30
Postleitzahl	8
Wohnort	30

Ferner ist zu klären, wie der Ausdruck auf den Adreßaufklebern, die auf Endlospapier aufgeklebt sind, erfolgen soll. Ein Adreßaufkleber umfaßt 8 Zeilen. Die beiden obersten und untersten Zeilen bleiben frei.



Zwischen zwei Adreßaufklebern besteht ein Zwischenraum von einer Zeile.

Datenanalyse

Einzugeben sind die Daten zu Name, Straße / Hausnummer, Postleitzahl und Wohnort. Ferner ist die Anzahl der Ausdrucke einzuspeichern.

```
Eingabedaten:  Name : string[40]
               Strasse: string[30]
               Postleitzahl: string[8]
```

Wohnort: string[30]
Anzahl: integer

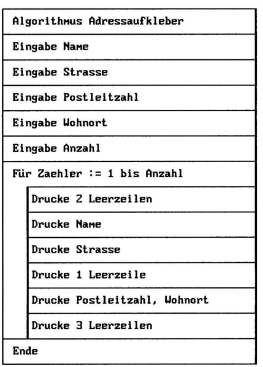
Auszugeben sind die eingegebenen Adreßdaten. Somit sind die Ausgabedaten gleich den Eingabedaten.

Lösungsweg (Algorithmus)

- Pseudocode

```
Beginn
    Eingabe Name, Strasse, Postleitzahl, Wohnort
    Eingabe Anzahl
    Für Zaehler := 1 bis Anzahl
        Drucke 2 Leerzeilen
        Drucke Name
        Drucke Strasse
        Drucke 1 Leerzeile
        Drucke Postleitzahl
        Drucke Wohnort
        Drucke 3 Leerzeilen
Ende
```

- **Struktogramm**



Programm

```
program Adressaufkleber;
uses crt, printer;                                    (* ab Version 4 *)
var Name: string [40];
    Strasse, Wohnort: string [30];
    Postleitzahl: string [8];
    Anzahl, Zaehler: integer;

begin
  ClrScr;
  write ('Name ...................:'); readln (Name);
  write ('Strasse - Hausnummer ...:'); readln (Strasse);
  write ('Postleitzahl............:'); readln (Postleitzahl);
  write ('Wohnort ................:'); readln (Wohnort);
  write ('Wieviele Drucke? .......:'); readln (Anzahl);
  for Zaehler := 1 to Anzahl do
    begin
      writeln(lst); writeln (lst);
      writeln (lst,Name);
      writeln (lst, Strasse);
      writeln (lst);
      writeln (lst,Postleitzahl,' ', Wohnort);
      writeln (lst); writeln (lst); writeln (lst);
    end;
end.
```

In der Ausgabeanweisung

```
writeln (lst,Postleitzahl,' ', Wohnort);
```

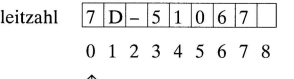
wurde ein Leerzeichen eingefügt. Dadurch soll erreicht werden, daß zwischen der Postleitzahl und dem Wohnort ein Zwischenraum von einem Zeichen gelassen wird. Ohne diese Leerzeichen-Konstante würde ausgedruckt:

```
D–51067Köln
```

Eine Zeichenkette wird stets in der wirklichen, effektiven Länge ausgegeben.

Sind in eine String-Variable 7 Zeichen eingegeben worden, so werden bei der Ausgabe auch nur 7 Zeichen wieder ausgegeben, auch wenn die String-Variable selber mit 8 Stellen vereinbart worden war. Die Anzahl der Zeichen, die ausgegeben wird, hängt also nicht von der Länge der String-Variablen ab. Es werden nur so viele Zeichen ausgegeben, wie in der String-Variablen enthalten sind.

Bei der Eingabe von Zeichen in eine String-Variable zählt der Rechner mit, wieviel Zeichen eingetastet wurden bis zur Betätigung der ↵-Taste. Die Zeichenanzahl speichert er in der String-Variablen in einem sogenannten Längenbyte.
Beim Ausdrucken oder Anzeigen einer String-Variablen werden so viele Zeichen ausgegeben, wie der Wert im Längenbyte angibt.



Für die Bearbeitung von Zeichenketten enthält Turbo-Pascal eine große Anzahl von vordefinierten Prozeduren und Funktionen. Sie sind im Kapitel 7 dargestellt.

6.2.1 Zusammenfassung

- Eine Folge von Zeichen bezeichnet man als Zeichenkette oder String.
- Der Datentypbezeichner für die Zeichenkette lautet **string**.
- Die maximale Länge einer Zeichenkettenvariablen beträgt 255 Stellen.
- Soll eine Stringvariable aus weniger als 255 Stellen bestehen, so ist neben der Datentypangabe string auch zu vermerken, aus wieviel Stellen die Zeichenkette maximal besteht. Version 3 läßt nur diese Form zu.

6.2.2 Übungen

1. Aufgabe
In einem Programm sind Artikelbezeichnungen zu verarbeiten. Die Länge der Artikelbezeichnungen bewegt sich zwischen 16 und 33 Zeichen. Mit welcher Längenangabe ist die String-Variable Art-Bezeichnung zu vereinbaren?

var ArtBezeichnung: _____

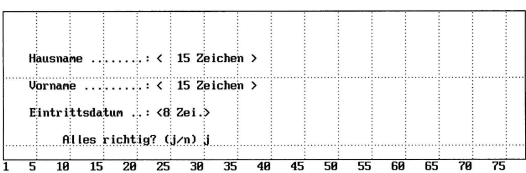
2. Aufgabe

Wie bewerkstelligt es der Rechner, daß Zeichenketten in ihrer wirklichen Länge ausgedruckt werden, auch wenn die String-Variable mehr Stellen aufweist?

3. Aufgabe

Entwickeln Sie ein Programm, mit dem Sie in komfortabler Weise eine Mitgliederliste erstellen können. In der Liste sollen Name, Vorname und Eintrittsdatum ausgewiesen werden.

Bildschirmaufbau



Der Programmablauf soll so gestaltet werden, daß die Angaben zunächst über die Tastatur eingegeben werden. Danach soll der Rechner fragen, ob die Daten korrekt sind. Falls nicht, wird die Eingabe wiederholt. Erst wenn die Korrektheitsabfrage mit j = ja beantwortet wird, soll die Ausgabe der Mitgliederdaten über den Drucker erfolgen. Falls Sie keinen Drucker zur Verfügung haben, steuern Sie die Ausgabe in die Zeile 20 des Bildschirms.
Nach dem Ausdrucken soll der Rechner fragen, ob noch weitere Daten eingegeben werden sollen. Wird diese Frage mit n = nein beantwortet, soll das Programm beendet werden.
Der nachstehende Pseudocode skizziert den Algorithmus:

```
Algorithmus Mitgliederliste
Beginn
  Aufbau Bildschirmmaske
  Wiederhole
    Wiederhole
      Eingabe Name
```

```
        Eingabe Vorname
        Eingabe Eintrittsdatum
        Eingabe Korrekt (j/n)
      bis Korrekt = 'j'
      Ausgabe Name, Vorname, Eintrittsdatum
      Eingabe Weiter (j/n)
    bis weiter = 'n'
  Ende
```

6.2.3 Lösungen

1. Aufgabe
Als Länge muß 33 vereinbart werden.

```
var ArtBezeichnung: string[33];
```

2. Aufgabe
Bei der Eingabe der Zeichenkette wird die Anzahl der eingegebenen Zeichen mitgezählt und in einem Längenbyte gespeichert. Beim Ausdruck oder bei der Anzeige auf dem Bildschirm werden nur so viele Zeichen ausgegeben, wie der Wert im Längenbyte angibt.

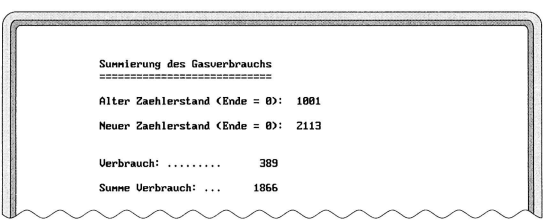
3. Aufgabe

```
program Mitgliederliste;
uses crt, printer;                                    (* ab Version 4 *)
var Name, Vorname: string[15];
    Eintrittsdatum: string[8];
    Korrekt, Weiter: char;

begin
  ClrScr;
  gotoxy(5,4); write ('Hausname .........: ');
  gotoxy(5,6); write ('Vorname ..........: ');
  gotoxy(5,8); write ('Eintrittsdatum ...: ');
  repeat
    repeat
      gotoxy (10,10); ClrEol;
      gotoxy (24, 4); ClrEol; readln (Name);
      gotoxy (24, 6); ClrEol; readln (Vorname);
      gotoxy (24, 8); ClrEol; readln (Eintrittsdatum);
      gotoxy (10,10); write ('Alles richtig? (j/n) ');
      readln (Korrekt);
    until Korrekt = 'j';
    writeln (lst, Name,' ',Vorname,' ',Eintrittsdatum);
    gotoxy (10,10); write ('Weitere Eingaben? (j/n) ');
    readln (weiter);
  until weiter = 'n';
end.
```

7 UNTERPROGRAMME

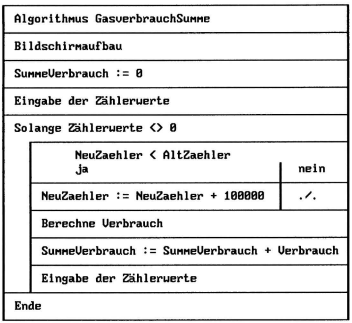
7.1 Benutzerdefinierte Prozeduren und Funktionen

Die Unterprogrammtechnik ist ein wichtiges Hilfsmittel zur Gestaltung von Pascal-Programmen. Das wird an Hand des Programms «GasverbrauchSumme», mit dem die Summe der verbrauchten Kubikmeter Gas ermittelt wird, dargelegt. Das Problem der Verbrauchsermittlung ist Ihnen bereits aus früheren Kapiteln bekannt.



Wie aus der Abbildung der Bildschirmmaske zu entnehmen ist, werden die Zählerstandswerte eingegeben. Daraus wird der Verbrauch berechnet. Der Verbrauch wird anschließend zur SummeVerbrauch aufgerechnet.
Dieser Vorgang wird so lange wiederholt, bis für die beiden Zählerwerte eine Null eingegeben wird.

Struktogramm



Programm

```
0   program GasverbrauchSumme;
    uses crt;                                        (* ab Version 4 *)
1   var AltZaehler, Neuzaehler, Verbrauch, SummeVerbrauch: real;
2   begin
3     ClrScr;
4     gotoxy (5, 3); write ('Summierung des Gasverbrauchs');
5     gotoxy (5, 4); write ('==========================');
6     gotoxy (5, 6); write ('Alter Zaehlerstand (Ende = 0): ');
7     gotoxy (5, 8); write ('Neuer Zaehlerstand (Ende = 0): ');
8     gotoxy (5,12); write ('Verbrauch: .........');
9     gotoxy (5,14); write ('Summe Verbrauch: ...');
10    SummeVerbrauch := 0;
11    gotoxy (37,6); ClrEol; readln (AltZaehler);
12    gotoxy (37,8); ClrEol; readln (NeuZaehler);
13    while (Altzaehler <> 0) or (NeuZaehler <> 0) do
14      begin
15        if NeuZaehler < AltZaehler
16          then NeuZaehler := NeuZaehler + 100000.0;
17        Verbrauch := Neuzaehler - Altzaehler;
18        SummeVerbrauch := SummeVerbrauch + Verbrauch;
19        gotoxy (27,12); write (Verbrauch:7:0);
20        gotoxy (27,14); write (SummeVerbrauch:7:0);
21        gotoxy (37,6); ClrEol; readln (AltZaehler);
22        gotoxy (37,8); ClrEol; readln (NeuZaehler);
23      end;
24    gotoxy (10,20); write ('Ende des Programms');
25  end.
```

An diesem Programm fällt auf, daß an zwei verschiedenen Stellen exakt dieselbe Folge von Anweisungen auftritt. Die Eingabeanweisungen

```
gotoxy (37,6); ClrEol; readln (AltZaehler); (Zeilen 11 und 21)
und
gotoxy (37,8); ClrEol; readln (NeuZaehler); (Zeilen 12 und 22)
```

mußten nämlich wegen des Prinzips des Erstlesens – Folgelesens sowohl vor der WHILE-Schleife als auch innerhalb der WHILE-Schleife programmiert werden.
Um den Programmieraufwand zu vermindern, bedient man sich in einem solchen Fall der Unterprogrammtechnik. In Pascal wird die Unterprogrammtechnik mittels Prozeduren und Funktionen verwirklicht. Wie das geschieht, wird in den nächsten Abschnitten dargelegt.

7.2 Einfache, parameterlose Prozeduren

Da die Eingabeanweisungen an zwei verschiedenen Stellen im Programm geschrieben werden mußten, ist es sinnvoll, diese Anweisungen zu einem Teilprogramm (Prozedur) zusammenzufassen und aus dem Programm «auszuklammern» beziehungsweise herauszuziehen.

Vereinbarung und Aufruf

```
procedure Zaehlereingabe;
begin
   gotoxy (37,6); ClrEol; readln (AltZaehler);
   gotoxy (37,8); ClrEol; readln (NeuZaehler);
end;
```

Die Prozedur wird eingeleitet mit dem Schlüsselwort **procedure**. Dahinter wird der Name der Prozedur angegeben. Hier lautet er: **Zaehlereingabe.** Die Kopfzeile der Prozedur wird durch ein Semikolon abgeschlossen.
Allgemeine Form der Prozedurvereinbarung:

```
procedure name;
```

Der Anweisungsteil der Prozedur beginnt mit einem **begin** und endet mit einem **end**. Der Prozeduraufbau entspricht also dem Aufbau eines Programms.

Unter Einbeziehung der Prozedur Zaehlereingabe lautet das Programm:

```
0  program GasverbrauchSumme;
   uses crt;                                    (* ab Version 4 *)
1  var AltZaehler, NeuZaehler, Verbrauch, SummeVerbrauch: real;

2  procedure Zaehlereingabe; (* Prozedurvereinbarung *)
3  begin
4    gotoxy (37,6); ClrEol; readln (AltZaehler);
5    gotoxy (37,8); ClrEol; readln (NeuZaehler);
6  end;

7  begin   (* Hauptprogramm *)
8    ClrScr;
9    gotoxy (5, 3); write ('Aufsummierung des Gasverbrauchs');
10   gotoxy (5, 4); write ('==============================');
11   gotoxy (5, 6); write ('Alter Zaehlerstand (Ende = 0): ');
12   gotoxy (5, 8); write ('Neuer Zaehlerstand (Ende = 0): ');
13   gotoxy (5,12); write ('Verbrauch: .........');
14   gotoxy (5,14); write ('Summe Verbrauch: ...');
15   SummeVerbrauch := 0;
16   Zaehlereingabe;
17   while (Altzaehler <> 0) or (NeuZaehler <> 0) do
18     begin
19       if NeuZaehler < AltZaehler
20         then NeuZaehler := NeuZaehler + 100000.0;
21         Verbrauch := Neuzaehler - Altzaehler;
22         SummeVerbrauch := SummeVerbrauch + Verbrauch;
23         gotoxy (27,12); write (Verbrauch:7:0);
24         gotoxy (27,14); write (SummeVerbrauch:7:0);
25         Zaehlereingabe;
26     end;
27   gotoxy (10,20); write ('Ende des Programms');
28 end.
```

Prozeduren werden zwischen dem Vereinbarungsteil und dem Anweisungsteil des Programms, jetzt auch Hauptprogramm genannt, vereinbart. In dem Hauptprogramm finden Sie an den Stellen, an denen vorher die Eingaberoutine gestanden hat, den Prozedurnamen **Zaehlereingabe** (Zeilen 16 und 25). Über den Prozedurnamen wird die Prozedur aus dem Hauptprogramm aufgerufen.

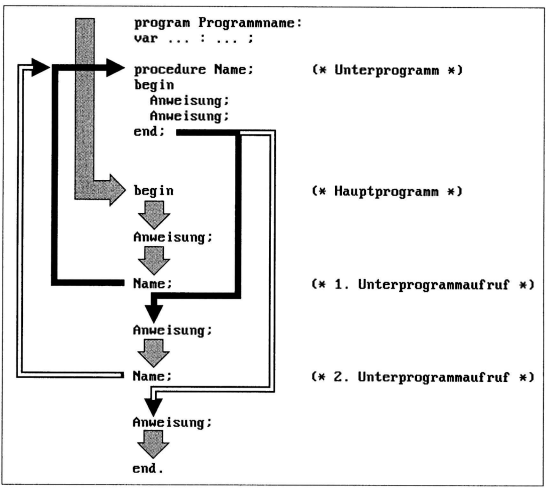
Ablaufsteuerung

Obwohl die Prozedurvereinbarung vor dem Hauptprogramm steht, wird sie beim Ablauf des Programms nicht vorweg behandelt. Beim

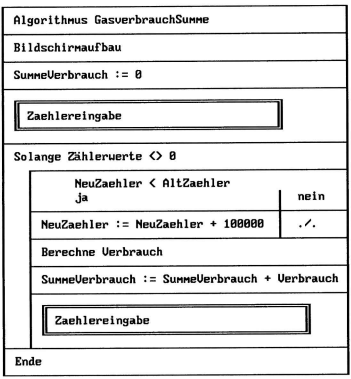
Starten des Programms werden die Anweisungen der Prozedur zunächst übergangen. Der erste Befehl, der in dem obigen Programm ausgeführt wird, ist der ClrScr-Befehl zum Löschen des Bildschirms in Zeile 8.

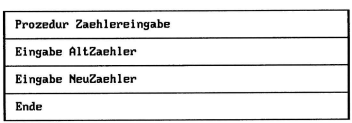
Erst an der im Hauptprogramm aufgerufenen Stelle wird die Prozedur bearbeitet. Im Anschluß an die Bearbeitung der Prozedur kehrt das System zum Hauptprogramm zurück und führt die Anweisung aus, die dem Prozeduraufruf folgt. In dem obigen Programmbeispiel wird jeweils nach der Prozedurbearbeitung die WHILE-Anweisung in Zeile 17 ausgeführt.

Die nachstehende Abbildung zeigt den Bearbeitungsablauf eines Programms mit Unterprogramm:



Im Struktogramm wird die Auslagerung der Eingabeanweisungen in eine Prozedur wie folgt zum Ausdruck gebracht:





In dem Struktogramm wird der Prozeduraufruf durch die Blockumrandung gekennzeichnet. Die Prozedur selbst wird als eigenständiges Struktogramm dargestellt.

Erweiterung des Programms

In dem obigen Programm GasverbrauchSumme sollen die eingegebenen Zählerstandswerte auf Plausibilität geprüft werden. Eine Zählerstandsangabe muß, wie sicherlich aus dem Einführungsbeispiel noch bekannt ist, kleiner als 100000 sein. Natürlich darf der Zählerstand auch nicht negativ sein. Er muß stets größer gleich null sein.

Wenn ein unzulässiger Wert eingegeben wird, soll in Zeile 20 der Fehlerhinweis

Eingabe unzulässig! <RETURN> drücken

erscheinen, der mit dem Drücken der ↵-Taste zu bestätigen ist. Anschließend soll der Fehlerhinweis gelöscht und die Eingabe wiederholt werden.

Die Eingabeprozedur lautet dann:

```
procedure Zaehlereingabe;          (* Prozedurvereinbarung *)
var Bestaetigung: char;
begin
  repeat
    gotoxy (37,6); ClrEol; readln (Alt2aehler);
    if (Alt2aehler < 0) or (Alt2aehler > 100000.0)
      then begin
             gotoxy (5,20);
             write ('Eingabe unzulässig! <RETURN> drücken');
             readln (Bestaetigung);
             gotoxy (5,20); DelLine;
           end;
  until (Alt2aehler >= 0) and (Alt2aehler < 100000.0);
  repeat
    gotoxy (37,8); ClrEol; readln (Neu2aehler);
    if (Neu2aehler < 0) or (Neu2aehler > 100000.0)
      then begin
             gotoxy (5,20);
             write ('Eingabe unzulässig! <RETURN> drücken');
             readln (Bestaetigung);
             gotoxy (5,20); DelLine;
           end;
  until (Neu2aehler >= 0) and (Neu2aehler < 100000.0);
end;
```

Diese Eingabeprozedur weist zwei neue Elemente auf.

DelLine

Zum einen wurde eine neue Anweisung (besser: Standardprozedur) eingeführt:

```
DelLine
```

Mittels **DelLine** wird die Bildschirmzeile, in der der Cursor steht, gelöscht. Alle nachfolgenden Zeilen werden um eine Zeile nach oben verschoben. Diese Wirkung kommt aber hier nicht zur Geltung, weil keine weitere Zeile folgt.

DelLine ist ein Prozedurname. Im Unterschied zu der obigen Prozedur Zaehlereingabe brauchen Sie aber nicht die Anweisungen, mit denen eine Zeile auf dem Bildschirm gelöscht werden kann, zu formulieren. Das ist bereits im Turbo-Pascal-System geschehen. Diese

Art von Prozedur, die bereits in das Sprachsystem «eingebaut» ist, wird als Standardprozedur bezeichnet.

Lokale und globale Variablen

Zum anderen wurde eine Variable Bestaetigung in der Prozedur vereinbart. Würden Sie die Variable Bestaetigung im Hauptprogramm ansprechen, würde der Compiler den Fehlerhinweis bringen, daß die Variable nicht bekannt ist. Der Grund dafür liegt darin, daß die Variable Bestaetigung innerhalb der Prozedur vereinbart worden ist. Damit ist sie nur gültig innerhalb der Prozedur, in der sie deklariert wurde. Sie wird darum als **lokale Variable** bezeichnet.

Eine Variable, die im Deklarationsteil des Hauptprogramms vereinbart wurde, kann sowohl im Hauptprogramm wie auch in einer Prozedur angesprochen werden.

Sie ist innerhalb aller Teile des Programms gültig. Eine solche Variable wird deshalb auch **globale Variable** genannt.
Das Verhältnis zwischen lokaler und globaler Variable soll noch näher beleuchtet werden. Beispielsweise wurde im Hauptprogramm eine Variable unter dem Namen Kosten vereinbart. Aber auch in einer Prozedur wird eine lokale Variable mit dem Namen Kosten deklariert. Trotz der Namensgleichheit sind sie für das System zwei verschiedene Variablen, für die unterschiedliche Speicherplätze bereitgestellt werden. Wenn die Prozedur gerade bearbeitet wird, so beziehen sich die Anweisungen, in denen die Variable Kosten angesprochen wird, auf den Speicherplatz der lokalen Variablen Kosten. Außerhalb der Prozedur Eingabe wird auf den Speicherplatz der globalen Variablen Kosten Bezug genommen.

Eingeschachtelte Prozedur

In der um die Fehlerbehandlung erweiterten Prozedur treten die Anweisungen

```
gotoxy (5,20);
write ('Eingabe unzulässig! <RETURN> drücken');
readln (Bestaetigung);
gotoxy (5,20); DelLine;
```

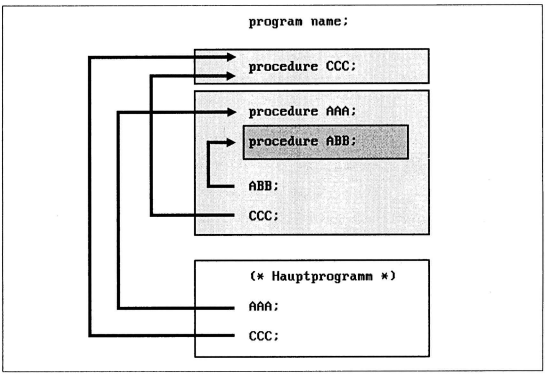
zweimal auf. Auch diese Anweisungen können Sie zu einer eigenen Prozedur innerhalb der Prozedur Zaehlereingabe zusammenfassen. Die Prozedur Zaehlereingabe hat mit der eingeschachtelten Prozedur Fehlermeldung folgende Form:

```
procedure Zaehlereingabe;          (* Prozedurvereinbarung *)

  procedure Fehlermeldung;         (* Prozedurvereinbarung *)
  var Bestaetigung: char;
  begin                            (* Beginn der Prozedur Fehlermeldung *)
    gotoxy (5,20);
    write ('Eingabe unzulässig! <RETURN> drücken');
    readln (Bestaetigung);
    gotoxy (5,20); DelLine;
  end;                             (* Ende der Prozedur Fehlermeldung *)

begin                              (* Beginn der Prozedur Zaehlereingabe *)
  repeat
    gotoxy (37,6); ClrEol; readln (AltZaehler);
    if (AltZaehler < 0) or (AltZaehler > 100000.0)
      then Fehlermeldung;
  until (AltZaehler >= 0) and (AltZaehler < 100000.0);
  repeat
    gotoxy (37,8); ClrEol; readln (NeuZaehler);
    if (NeuZaehler < 0) or (NeuZaehler > 100000.0)
      then Fehlermeldung;
  until (NeuZaehler >= 0) and (NeuZaehler < 100000.0);
end;                               (* Ende der Prozedur Zaehlereingabe *)
```

Wird innerhalb einer Prozedur eine «Unter»-Prozedur vereinbart, so ist die Vereinbarung der «Unter»-Prozedur zwischen dem Prozedurkopf und dem Anweisungsteil der Prozedur einzuschieben.

Die «Unter»-Prozedur kann nur aus der Prozedur aufgerufen werden, innerhalb der sie vereinbart wurde. So ist es beispielsweise nicht möglich, die Prozedur Fehlermeldung aus dem Hauptprogramm aufzurufen.

Falls eine Prozedur aus verschiedenen Prozeduren und/oder aus dem Hauptprogramm aufgerufen werden soll, muß sie als selbständige Prozedur vereinbart werden. Auch hier gilt die Regel, daß im Programm eine Prozedur vor ihrem ersten Aufruf vereinbart worden sein muß.



Die Beschreibung des Aufbaus einer Prozedur und ihre Einbindung ist also noch zu erweitern. Wie das obige Beispiel zeigt, können Sie für eine Prozedur lokale Variablen vereinbaren. Ferner können Sie für die Prozedur eigene Konstanten sowie auch «Unter»-Prozeduren deklarieren.
In verallgemeinerter Form sieht der Aufbau eines Programms unter Einbeziehung von Prozeduren so aus:

```
program ProgrammName;
var...                    (* Vereinbarung der globlen Variablen *)
procedure Name;
var...                    (* Vereinbarung von lokalen Variablen *)
   procedure Unter;
   var...                 (* Vereinbarung von lokalen Variablen *)
   begin (* Unter *)
      :                   (* Anweisungsteil von Prozedur Unter *)
      .
   end;  (* Unter *)
begin    (* Name *)
   :                      (* Anweisungsteil von Prozedur Name *)
   .
end;      (* Name *)
```

```
begin    (* Hauptprogramm *)
  :                (* Anweisungsteil des Hauptprogramms *)

.
end.     (* Hauptprogramm *)
```

Wie das obige Beispiel der Prozedur Zaehlereingabe mit der eingeschachtelten Prozedur Fehlermeldung zeigt, vermindert sich durch Verwendung von Prozeduren zum einen der Programmier- und Schreibaufwand. Stellen Sie sich einmal vor, wie lang das Programm «GasverbrauchSumme» sein würde, wenn auch die Fehlerbehandlung im Hauptprogramm programmiert worden wäre.
Zum anderen gewinnt ein Programm unter Einbeziehung von Prozeduren an Übersichtlichkeit. Mittels Prozeduren können Sie Detailprobleme aus dem Gesamtproblem herausziehen und Ihre Aufmerksamkeit ganz der Lösung des Detailproblems widmen.

7.2.1 Zusammenfassung

- Prozeduren vermindern den Programmieraufwand und erhöhen die Übersichtlichkeit des Programms.
- Prozeduren sind vor dem Anweisungsteil des Hauptprogramms beziehungsweise vor ihrem ersten Aufruf zu vereinbaren.
- Innerhalb einer Prozedur können weitere Prozeduren eingeschachtelt werden.
- Variablen, die im Vereinbarungsteil der Prozedur vereinbart werden, nennt man **lokale Variablen**. Ihr Gültigkeitsbereich ist auf die Prozedur beschränkt, in der sie deklariert wurden.

7.2.2 Übungen

1. Aufgabe
Welcher Unterschied besteht zwischen einer lokalen und einer globalen Variablen?

2. Aufgabe
Gestalten Sie das obige Programm GasverbrauchSumme in der Weise um, daß die Anweisungen zum Bildschirmaufbau in einer Prozedur zusammengefaßt werden.

3. Aufgabe

Mit einem Programm sollen einzelne Kostenbeträge summiert werden. Dabei gelten folgende Verarbeitungsbedingungen:

(1) Negative Kostenbeträge sind nicht zulässig. Wenn ein negativer Betrag eingegeben wird, muß die Eingabe mit einem Fehlerhinweis abgewiesen werden.

(2) Die Eingabe der Kostenbeträge soll beendet werden, wenn als Kostenbetrag eine Null eingegeben wird.

Gestalten Sie das folgende Programm durch Verwendung der Prozedurtechnik um.

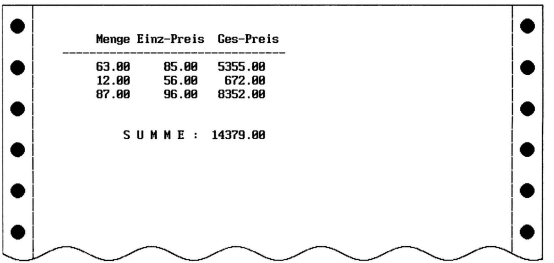
```
program Kostenermittlung;
uses crt;                                          (* ab Version 4 *)
var Kosten, SummeKosten: real;
begin
  ClrScr;
  SummeKosten := 0;
  repeat
    gotoxy(5,10);
    write ('Kosten: (0 = Ende) ..: '); ClrEol; readln (Kosten);
    if Kosten < 0
      then begin
             write (' Fehler! Eingabe wiederholen!');
             delay(800); DelLine;
           end;
  until Kosten >= 0;
  while Kosten <> 0 do
    begin
      SummeKosten := SummeKosten + Kosten;
      repeat
        gotoxy(5,10);
        write ('Kosten: (0 = Ende) ..: '); ClrEol; readln (Kosten);
        if Kosten < 0
          then begin
                 write (' Fehler! Eingabe wiederholen!');
                 delay(800); DelLine;
               end;
      until Kosten >= 0;
    end;
  gotoxy (5,12);
  writeln ('Die Summe der Kosten betraegt',SummeKosten:10:2);
end.
```

4. Aufgabe

Entwickeln Sie ein Programm, mit dem der Lagerwert ermittelt werden kann.

Eingaben: Menge
Einzelpreis

Ausgabe über Drucker:



```
Menge Einz-Preis  Ges-Preis
-------------------------------
63.00     85.00   5355.00
12.00     56.00    672.00
87.00     96.00   8352.00

    S U M M E :  14379.00
```

Hinweise zur Programmgestaltung:
(1) Die Eingabe von Menge und Einzelpreis soll in Form einer einfachen Prozedur gestaltet werden.
(2) Die Überschrift soll als einfache Prozedur geschrieben werden.
(3) Das Programm ist als Schleifenkonstruktion auszulegen, die so lange durchlaufen wird, bis für Menge eine Null eingegeben wird.

7.2.3 Lösungen

1. Aufgabe
Eine lokale Variable gilt nur in der Prozedur bzw. Funktion, in der sie vereinbart wurde. Globale Variablen können in jeder Prozedur / Funktion sowie im Hauptprogramm angesprochen werden.

2. Aufgabe

Das Programm lautet:

```
program GasverbrauchSumme;
uses crt;                                              (* ab Version 4 *)
var AltZaehler, NeuZaehler, Verbrauch, SummeVerbrauch: real;

  procedure Zaehlereingabe;    (* Prozedurvereinbarung *)
    procedure Fehlermeldung;   (* 'Unter'-Prozedurvereinbarung *)
    var Bestaetigung: char;    (* lokale Variable *)
    begin                      (* Anweisungsteil von Fehlermeldung *)
      gotoxy (5,20);
      write ('Eingabe unzulässig! <RETURN> drücken');
      readln (Bestaetigung);
      gotoxy (5,20); DelLine;
    end;

  begin                        (* Anweisungsteil von Zaehlereingabe *)
    repeat
      gotoxy (37,6); ClrEol; readln (AltZaehler);
      if (AltZaehler < 0) or (AltZaehler > 100000.0)
        then Fehlermeldung;
    until (AltZaehler >= 0) and (AltZaehler < 100000.0);
    repeat
      gotoxy (37,8); ClrEol; readln (NeuZaehler);
      if (NeuZaehler < 0) or (NeuZaehler > 100000.0)
        then Fehlermeldung;
    until (NeuZaehler >= 0) and (NeuZaehler < 100000.0);
  end;

  procedure Bildschirmmaske;       (* Prozedurvereinbarung *)
  begin
    ClrScr;
    gotoxy (5, 3); write ('Aufsummierung des Gasverbrauchs');
    gotoxy (5, 4); write ('==============================');
    gotoxy (5, 6); write ('Alter Zaehlerstand (Ende = 0): ');
    gotoxy (5, 8); write ('Neuer Zaehlerstand (Ende = 0): ');
    gotoxy (5,12); write ('Verbrauch: .........');
    gotoxy (5,14); write ('Summe Verbrauch: ...');
  end;

begin                              (* Hauptprogramm *)
  Bildschirmmaske;
  SummeVerbrauch := 0;
  Zaehlereingabe;
  while (Altzaehler <> 0) or (NeuZaehler <> 0) do
    begin
      if NeuZaehler < AltZaehler
        then NeuZaehler := NeuZaehler + 100000.0;
      Verbrauch := Neuzaehler - Altzaehler;
      SummeVerbrauch := SummeVerbrauch + Verbrauch;
      gotoxy (27,12); write (Verbrauch:7:0);
      gotoxy (27,14); write (SummeVerbrauch:7:0);
      Zaehlereingabe;
    end;
  gotoxy (10,20); write ('Ende des Programms');
end.
```

3. Aufgabe

```
program Kostenermittlung;
uses crt;                                              (* ab Version 4 *)
var Kosten, SummeKosten : real;

  procedure Eingabe;
  begin
    repeat
      gotoxy(5,10);
      write ('Kosten: (0 = Ende) ..: '); ClrEol; readln (Kosten);
      if Kosten < 0
        then begin
               write (' Fehler! Eingabe wiederholen');
               delay (800); DelLine;
             end;
    until Kosten >= 0;
  end;

begin   (* Hauptprogramm *)
  ClrScr;
  SummeKosten := 0;
  Eingabe;  (* Prozeduraufruf *)
  while Kosten <> 0 do
    begin
      SummeKosten := SummeKosten + Kosten;
      Eingabe;   (* Prozeduraufruf *)
    end;
  gotoxy (5,12);
  writeln ('Die Summe der Kosten betraegt',SummeKosten:10:2);
end.
```

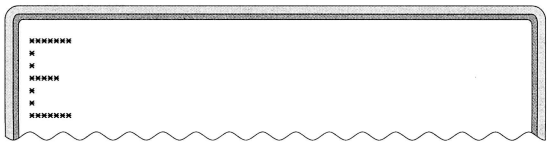
4. Aufgabe

```
program LagerSumme;
uses crt, printer;                                    (* ab Version 4 *)
var Menge, Einzelpreis, Gesamtpreis, Summe: real;

  procedure Eingabe;
  begin
    write ('Eingabe Menge (Ende = 0): '); readln (Menge);
    if Menge > 0
      then begin
        write ('Eingabe Einzelpreis     : '); readln (Einzelpreis);
      end;
  end;

  procedure Ueberschrift;
  begin
    writeln (lst,'     Menge Einz-Preis  Ges-Preis');
    writeln (lst,'-----------------------------------');
  end;

begin (* Hauptprogramm *)
  ClrScr;
  Summe := 0;
  Ueberschrift;
  Eingabe;
  while Menge > 0 do
    begin
      Gesamtpreis := Menge * Einzelpreis;
      writeln (lst, Menge: 10:2, Einzelpreis:10:2, Gesamtpreis:10:2);
      Summe := Summe + Gesamtpreis;
      Eingabe;
    end;
  writeln (lst); writeln (lst);
  writeln (lst, '         S U M M E :',Summe:10:2);
end.
```

7.3 Prozedur mit Wertübergabe

Die Prozedur mit Wertübergabe sei an einem kleinen Beispielprogramm erläutert.

Programmbeispiel **Großbuchstabe E**
Es soll der Buchstabe E mit Hilfe von Sternchen in Großformat geschrieben werden:



Das erreichen Sie dadurch, daß Sie in der ersten Zeile 7 Sternchen, in den beiden folgenden jeweils ein Sternchen auf dem Bildschirm anzeigen lassen. In der 4. Zeile werden 5 Sternchen ausgegeben. Daran schließt sich die Ausgabe von jeweils einem Sternchen in den nächsten beiden Zeilen an. In der letzten Zeile schließlich sind wieder 7 Sternchen anzuzeigen.
Pro Zeile ist also eine unterschiedliche Anzahl von Sternchen auszugeben. Das Programm dazu lautet:

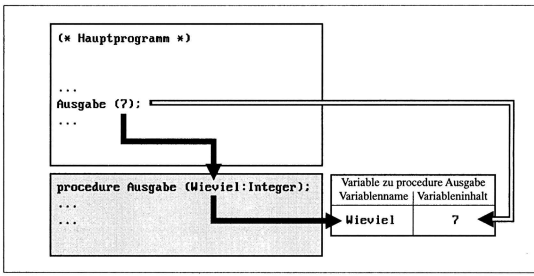
```
 0  program Buchstabe_E;
    uses crt;                                    (* ab Version 4 *)
 1    procedure Ausgabe (Wieviel: integer);
 2    var Zaehler : integer;
 3    begin
 4      for Zaehler := 1 to Wieviel do write ('*');
 5      writeln;
 6    end;

 7  begin (* Hauptprogramm *)
 8    ClrScr;
 9    Ausgabe (7);
10    Ausgabe (1);
11    Ausgabe (1); Ausgabe (5);
12    Ausgabe (1); Ausgabe (1); Ausgabe (7);
13  end.
```

Die Prozedur, über die die Ausgabe der Sternchen erfolgt, wurde mit dem Namen Ausgabe bezeichnet. Die Prozedur **Ausgabe** wird aus dem Hauptprogramm aufgerufen durch Nennung des Prozedurnamens, gefolgt von einer eingeklammerten Ziffer.

Ablaufsteuerung
Das Programm startet in Zeile 8 mit der Standardprozedur ClrScr, mit der der Bildschirm gelöscht wird. Als nächstes wird der Prozeduraufruf

Ausgabe (7)

bearbeitet. Die in Klammern stehende Zahl 7 wird dabei an die Prozedur übergeben. Beim Aufruf der Prozedur wird eine Integer-Variable mit dem Namen Wieviel angelegt, der die Zahl 7 zugewiesen wird.



In dem Anweisungsteil der Prozedur werden nun so viele Sternchen ausgegeben, wie in der lokalen Variablen Wieviel angegeben ist. Nachdem eine Zeilenschaltung in Zeile 5 ausgeführt ist, wird die Prozedur mit einem Rücksprung ins Hauptprogramm verlassen. Als nächste Anweisung wird in Zeile 10 der Prozeduraufruf

Ausgabe (1)

ausgeführt. Dabei wird die Zahl 1 an die Prozedur Ausgabe übergeben. Die Prozedur Ausgabe läuft dann in der gerade beschriebenen Weise ab.

Aktuelle und formale Parameter

Im Gegensatz zu der oben besprochenen einfachen Prozedur weist die Prozedur Ausgabe einen Zusatz auf, der in runden Klammern hinter dem Prozedurnamen vermerkt ist:

(Wieviel: integer);

Der Zusatz bedeutet, daß beim Aufruf der Prozedur eine ganzzahlige Variable einzurichten ist, die den Namen Wieviel erhält. Dieser Zusatz wird auch als **formaler Parameter** bezeichnet.
Die als formaler Parameter angegebene Variable Wieviel dient dazu, den Wert aufzunehmen, der in runden Klammern hinter dem Prozeduraufruf steht. Die Erweiterung des Prozeduraufrufs, hier beispielsweise Ausgabe (7), nennt man **aktuellen Parameter**.
Weil beim Prozeduraufruf ein Wert an die Prozedur übergeben wird, handelt es sich hier um eine Prozedur mit Wertübergabe. Auch ist der Begriff einer Prozedur mit **Werteparameter** gebräuchlich.

Vereinbarung und Aufruf
Allgemein lautet der Prozeduraufruf:

```
Prozedurname (Liste der aktuellen Parameter)
```

Aktuelle Parameter können sein:
- Konstanten
- Variablen, deren Inhalt an die Prozedur übergeben wird
- Ausdrücke, deren Wert an die Prozedur übergeben wird

Die Liste der aktuellen Parameter kann aus einem oder mehreren Elementen bestehen. Werden mehrere Elemente angegeben, so sind sie durch ein Komma voneinander zu trennen.
Beispiel eines Prozeduraufrufs mit mehreren Elementen:

```
Drucke (Menge, Preis, Preis*Menge);
```

In diesem Beispiel wird an eine Prozedur Drucke der Inhalt der Variablen Menge und Preis sowie der Wert der Multiplikation von Preis*Menge übergeben.
Die Vereinbarung einer **Prozedur mit Wertübergabe lautet** allgemein:

```
procedure Prozedurname (Liste der formalen Parameter);
```

In der Liste der formalen Parameter werden die Variablen aufgeführt, die die übergebenen Werte aufnehmen. Zu jeder Variablen ist ein Datentyp anzugeben, damit ein entsprechender Speicherplatz für

die Übernahme der Werte aus dem Prozeduraufruf bereitgestellt werden kann. Das sei am Beispiel der Vereinbarung einer Prozedur Drucke dargelegt, die durch

```
Drucke (Menge, Preis, Preis*Menge);
```

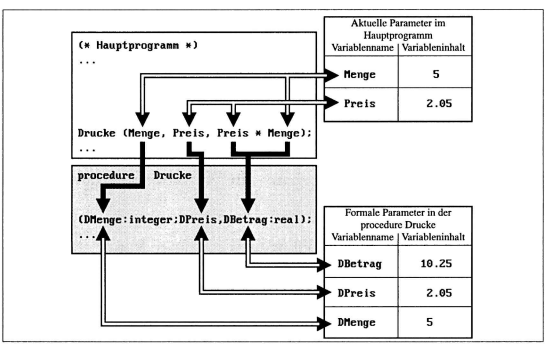
aufgerufen wird.
Die Variable Menge sei vom Typ Integer, Preis sei vom Typ Real. Damit in der Prozedur Drucke die Werte übernommen werden können, müssen im Kopf der Prozedur Drucke als formale Parameter vereinbart werden:

```
procedure Drucke (DMenge: integer; DPreis, DBetrag: real);
```

Beim Aufruf der Prozedur werden folgende Wertübertragungen vorgenommen:

```
DMenge := Menge
DPreis := Preis
DBetrag := Preis*Menge
```

Das veranschaulicht auch folgende Abbildung:



Die Wertübertragung erfolgt systematisch der Reihe nach.

1. formaler Parameter := 1. aktueller Parameter
2. formaler Parameter := 2. aktueller Parameter usw.

Deshalb ist es wichtig, daß die **aktuelle und die formale Parameterliste hinsichtlich Anzahl und Datentyp der Elemente übereinstimmen.** Wäre beispielsweise DBetrag als Integer-Variable vereinbart worden, so würde der Rechner die mangelnde Typübereinstimmung beanstanden. Bestünde die Liste der aktuellen Parameter nur aus zwei statt aus drei Elementen, so würde der Rechner den Fehlerhinweis geben, daß die Anzahl der Parameter nicht übereinstimmt.
Zu jeder Variablen in der formalen Parameterliste ist der Datentyp anzugeben. Das bedeutet aber nicht, daß hinter jeder Variablen unmittelbar die Angabe des Datentyps folgen muß. Vielmehr können Variablen gleichen Typs in einer Auflistung, die durch einen Doppelpunkt abgeschlossen wird, zusammengefaßt werden. Hinter dem Doppelpunkt ist der Datentyp anzugeben.

Beispiel:

```
procedure Beispiel (A, B, C: integer);
```

Folgen noch weitere Vereinbarungen innerhalb der formalen Parameterliste, so ist die folgende Vereinbarung von der vorherigen durch ein Semikolon zu trennen.

Beispiel:

```
procedure Beispiel (A, B, C: integer;X, Y: real; Zeich:char);
```

Hinsichtlich der Angabe des Datentyps ist die Vorschrift zu beachten, daß die Datentypangabe nur aus einem einfachen Wort (Bezeichner) bestehen darf. Wenn Sie beispielsweise folgendes Programm eingeben und übersetzen lassen, meldet der Compiler «Type Bezeichner erwartet»:

```
program Bezeichner_Test;
var Eingabetext: string [80];

procedure Ausgabe (Ausgabetext: string[80]);
begin
  writeln (Ausgabetext);
end;

begin                                    (* Hauptprogramm *);
  readln (Eingabetext);
  Ausgabe (Eingabetext);
end.
```

Wenn Sie nach dem Erscheinen der Fehlermeldung Esc drücken, steht der Cursor im Prozedurkopf auf string[80]. Diese Datentypangabe besteht nämlich nicht nur aus einem einfachen Wort, sondern ist vielmehr aus dem Wort string und der Längenangabe [80] zusammengesetzt.

Type-Vereinbarung

Dieses Problem kann man umgehen, indem man sich einen eigenen Datentyp schafft. Man legt in einer sogenannten Type-Vereinbarung zum Beispiel fest, daß der Datentyp string[80] mit dem Namen LangerText bezeichnet werden soll. Mit der Type-Vereinbarung

```
type LangerText = string[80]
```

wird der Datentyp LangerText vereinbart, der mit dem Datentyp string[80] identisch ist.
Die Type-Vereinbarung erfolgt im Deklarationsteil des Programms. Das obige Programm Bezeichner_Test ist dann wie folgt zu formulieren:

```
program Bezeichner_Test;
type LangerText = string [80];
var Eingabetext : string [80];

procedure Ausgabe (Ausgabetext: LangerText);
begin
  writeln (Ausgabetext);
end;

begin                                    (* Hauptprogramm *);
  readln (Eingabetext);
  Ausgabe (Eingabetext);
end.
```

Ändern Sie Ihr eingegebenes Programm entsprechend ab, und testen Sie es.

Falls Sie es noch nicht gespeichert haben, sollten Sie es tun. Auf dieses Programm wird später noch einmal Bezug genommen.

Die Angabe des Datentyps von Ausgabetext besteht nun aus einem einfachen Bezeichner, wenngleich sich dahinter eine komplexe Datenstruktur verbirgt.

Für den Compiler ist es nicht wichtig, in welcher Weise sich eine Datenstruktur zusammensetzt. Entscheidend ist für ihn, daß der Datentyp mit einem einfachen Wort bezeichnet wird.

Natürlich hätten Sie auch die Type-Vereinbarung LangerText in der Vereinbarung der Variablen Eingabetext verwenden können:

```
var Eingabetext: LangerText;
```

Mit der Type-Vereinbarung können Sie sich eigene Datentypen schaffen. Von dieser Möglichkeit wird an anderer Stelle intensiv Gebrauch gemacht.

7.3.1 Zusammenfassung

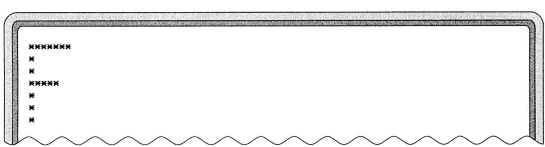
- Bei einer Prozedur mit Werteparametern können Werte an die Prozedur übergeben werden.
- Die zu übergebenden Werte werden beim Prozeduraufruf als aktuelle Parameter angegeben.
- Zur Übernahme der Werte werden in der Prozedurvereinbarung Variablen als formale Parameter vereinbart.
- Die aktuellen und formalen Parameter müssen hinsichtlich Anzahl und Datentyp übereinstimmen.
- Bei der Angabe des Datentyps eines formalen Parameters darf nur ein einfacher Bezeichner (Name) verwendet werden. Besteht der Datentypbezeichner aus mehreren Bestandteilen (zum Beispiel Typname + Längenangabe), so muß für diesen Datentyp mittels einer Typ-Vereinbarung ein eigener Datentyp geschaffen werden.

7.3.2 Übungen

1. Welcher Zusammenhang besteht zwischen aktuellen und formalen Parametern?
2. Wo werden die formalen Parameter vereinbart?
3. Welche Besonderheit ist hinsichtlich der Angabe des Datentyps bei den formalen Parametern zu beachten?
4. Wie muß die Liste der formalen Parameter beschaffen sein, wenn folgende Daten an eine Prozedur übergeben werden sollen?

```
program Probe;
var Menge: integer; Preis: real;
procedure Testup (_____);
begin... end;
begin (* Hauptprogramm *)
Testup (Menge,Preis);
...
```

5. Schreiben Sie ein Programm, mit dem der Buchstabe



auf dem Bildschirm erzeugt wird.

7.3.3 Lösungen

1. Aktuelle Parameter werden beim Aufruf der Prozedur angegeben. Sie beinhalten die Werte, die an die Prozedur bzw. Funktion übergeben werden. Im Kopf der Prozedur müssen entsprechende formale Parameter bereitgestellt werden.
2. Formale Parameter werden im Prozedurenkopf vereinbart.
3. Die Angabe des Datentyps muß aus einem einzigen einfachen

Wort bestehen. Bei zusammengesetzten Datentypnamen, zum Beispiel string [80], muß vorher mittels der Type-Anweisung ein eigener Datentyp geschaffen werden.
Beispiel:

```
type Text80 = string[80];
...
procedure irgendwas(Textzeile: Text80);
```

4. Die formalen Parameter sind:

```
procedure Testup (TMenge: integer; TPreis : real);
```

Welche Namen für die Variablen vergeben werden, ist nicht entscheidend. Entscheidend ist, daß zwei Variablen angegeben werden, von denen die erste den Datentyp Integer und die zweite den Datentyp Real aufweisen muß.

5. Das Programm lautet:

```
program Buchstabe_F;
uses crt;                                          (* ab Version 4 *)

  procedure Ausgabe (x: integer);
  var Zaehler : integer;
  begin
    for Zaehler := 1 to x do write ('*');
    writeln;
  end;

begin                                        (* Hauptprogramm *)
  ClrScr;
  Ausgabe (7);
  Ausgabe (1);
  Ausgabe (1);
  Ausgabe (5);
  Ausgabe (1);
  Ausgabe (1);
  Ausgabe (1);
end.
```

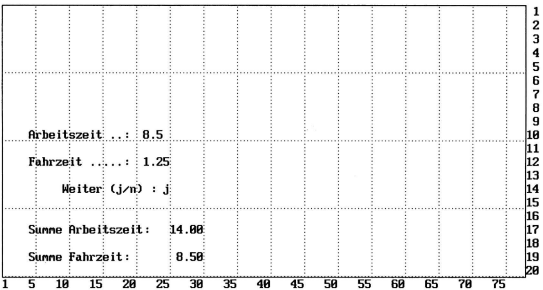
7.4 Prozedur mit Variablenübergabe

Prozeduren mit Werteparametern ermöglichen es, gleichlautende Folgen von Anweisungen, die an verschiedenen Stellen des Programms mit unterschiedlichen Werten arbeiten, zu einer Prozedur zusammenzufassen. Das Prozedurenkonzept von Pascal ist aber noch umfangreicher. Das zeigt folgendes Beispiel.

***Programmbeispiel* Dienstzeit**
In einem Unternehmen soll ermittelt werden, wieviel Stunden «echter» Arbeitszeit von den Außendienstmitarbeitern geleistet werden und wieviel Zeit sie für die Fahrten vom Betrieb zu den Kunden aufwenden. Die Stundenangaben zur Arbeitszeit und Fahrzeit werden an Hand der Abrechnungsbelege der Außendienstmitarbeiter in den Rechner eingegeben.

Kundendienstbeleg	
Kunde Albert Summer KG	
Wohnort Essen	
Telefon 0201 / 12345	
Ausgeführte Arbeit Schalter ausgebaut und erneuert	
Arbeitszeit 8,5	**Datum** 11.7.
Fahrzeit 1,25	**Unterschrift** A Summer

Die eingegebenen Werte werden summiert und am Bildschirm angezeigt. Die Verarbeitung soll so lange wiederholt werden, bis auf die Weiter-Abfrage mit «n» geantwortet wird.



Die Eingabe der Stunden ist auf Plausibilität zu prüfen. Die Stundenwerte müssen stets größer als oder gleich 0 und kleiner als oder gleich 10 sein. Wenn ein Stundenwert nicht innerhalb dieser Grenzen liegt, ist seine Eingabe zu wiederholen.
Der Algorithmus hat folgende Grobstruktur:

```
Algorithmus Dienstzeit
Bildschirmaufbau
SummeArbeitszeit := 0 , SummeFahrzeit := 0
    Eingabe Arbeitszeit
  bis Arbeitszeit >= 0 und Arbeitszeit <= 10
    Eingabe Fahrzeit
  bis Fahrzeit >= 0 und Fahrzeit <= 10
  SummeArbeitszeit := SummeArbeitszeit + Arbeitszeit
  Ausgabe SummeArbeitszeit
  SummeFahrzeit := SummeFahrzeit + Fahrzeit
  Ausgabe SummeFahrzeit
  Eingabe Weiter (j/n)
bis Weiter = 'n'
Ende
```

Das Pascal-Programm zur Ermittlung der Arbeitszeitsumme und Fahrzeitsumme lautet:

```
0  program Dienstzeit;
   uses crt;                                          (* ab Version 4 *)
1  var Arbeitszeit, Fahrzeit, SummeArbeitszeit, SummeFahrzeit: real;
2      Weiter: char;
3  begin
4    ClrScr;
5    gotoxy ( 5,10); write ('Arbeitszeit ..: ');
6    gotoxy ( 5,12); write ('Fahrzeit .....: ');
7    gotoxy (10,14); write ('Weiter (j/n) : ');
8    gotoxy ( 5,17); write ('Summe Arbeitszeit:');
9    gotoxy ( 5,19); write ('Summe Fahrzeit:   ');
10   SummeFahrzeit := 0; SummeArbeitszeit := 0;
11   repeat
12     repeat
13       gotoxy (22,10); ClrEol; readln (Arbeitszeit);
14     until (Arbeitszeit >= 0) and (Arbeitszeit <= 10);
15     repeat
16       gotoxy (22,12); ClrEol; readln (Fahrzeit);
17     until (Fahrzeit >= 0) and (Fahrzeit <= 10);
18     SummeArbeitszeit := SummeArbeitszeit + Arbeitszeit;
19     gotoxy (22,17); write (SummeArbeitszeit:8:2);
20     SummeFahrzeit := SummeFahrzeit + Fahrzeit;
21     gotoxy (22,19); write (SummeFahrzeit:8:2);
22     gotoxy (25,14); read (Weiter);
23   until Weiter = 'n';
24 end.
```

Die Eingabe der beiden Zeitangaben wird in den Zeilen 12 bis 14 sowie in den Zeilen 15 bis 17 vorgenommen. Die beiden Anweisungsfolgen sind in gleicher Weise aufgebaut. Die einzigen Unterschiede bestehen darin, daß sich die Anweisungen auf unterschiedliche Variablen beziehen und daß unterschiedliche Bildschirmpositionen angesprochen werden. Das Problem der unterschiedlichen Bildschirmpositionen wird im Augenblick noch außer acht gelassen.

Auch solche Folgen von Anweisungen, die in ihrem Aufbau gleichartig sind, sich jedoch auf unterschiedliche Variablen beziehen, kann man in Pascal zu einer Prozedur zusammenfassen.

Unter Einbeziehung einer solchen Prozedur lautet das Programm:

```
0  program Dienstzeit_2;
   uses crt;                                          (* ab Version 4 *)
1  var Arbeitszeit, Fahrzeit, SummeArbeitszeit, SummeFahrzeit: real;
2      Weiter: char;

3    procedure Zeiteingabe (var Zeit: real);
4    begin
5      repeat
6        ClrEol;
7        readln (Zeit);
8      until (Zeit >= 0) and (Zeit <= 10);
9    end;

10 begin   (* Hauptprogramm *)
11   ClrScr;
12   gotoxy (5,10); write ('Arbeitszeit ..: ');
13   gotoxy (5,12); write ('Fahrzeit .....: ');
14   gotoxy (10,14); write ('Weiter (j/n) : ');
15   gotoxy (5,17); write ('Summe Arbeitszeit:');
16   gotoxy (5,19); write ('Summe Fahrzeit:   ');
17   SummeFahrzeit := 0; SummeArbeitszeit := 0;
18   repeat
19     gotoxy (22,10);
20     Zeiteingabe (Arbeitszeit);
21     gotoxy (22,12);
22     Zeiteingabe (Fahrzeit);
23     SummeArbeitszeit := SummeArbeitszeit + Arbeitszeit;
24     gotoxy (23,17); write (SummeArbeitszeit:8:2);
25     SummeFahrzeit := SummeFahrzeit + Fahrzeit;
26     gotoxy (23,19); write (SummeFahrzeit:8:2);
27     gotoxy (25,14); read (Weiter);
28   until Weiter = 'n';
29 end.
```

In diesem Programm ist die Eingabe der Zeit sowie die Plausibilitätsprüfung in der Prozedur **Zeiteingabe** (Zeilen 3 bis 9) zusammengefaßt worden. Die Prozedur wird aus dem Hauptprogramm aufgerufen über

Zeiteingabe (Arbeitszeit)	in Zeile 20
beziehungsweise	
Zeiteingabe (Fahrzeit)	in Zeile 22

Bei diesem Aufruf wird jedoch nicht der jeweilige Wert der Variablen Arbeitszeit beziehungsweise Fahrzeit, sondern es wird die **Variable** selbst an die Prozedur übergeben. Weil an die Prozedur Zeiteingabe eine Variable übergeben wird, charakterisiert man sie als Prozedur mit **Variablenparameter**.

Vereinbarung und Aufruf

Der Prozeduraufruf mit Variablenübergabe hat denselben Aufbau wie der Prozeduraufruf mit Wertübergabe (Werteparameter): Nach dem Prozedurnamen folgt in runden Klammern die Angabe der aktuellen Parameter. Als aktuelle Parameter sind jedoch nur Variablen zulässig.

Allgemeine Form des **Prozeduraufrufs mit Variablenparameter**:

```
Prozedurname (Variable, Variable, ...)
             \_____aktuelle Parameter_____/
```

Bei der Prozedurvereinbarung wird im Kopf der Prozedur angegeben, daß nicht Werte, sondern Variablen an die Prozedur übergeben werden soll.

```
procedure Zeiteingabe (var Zeit: real);
```

Durch das Schlüsselwort **var**, das dem formalen Parameter Zeit vorangestellt ist, wird Zeit zum Variablenparameter.

Allgemeine Form der Prozedurvereinbarung mit Variablenparameter:

```
procedure Prozedurname (var Variable, Variable, ... : datentyp);
                       \_____formale Parameter_____/
```

Die Beziehungen zwischen den aktuellen und formalen Parametern sind bei der Prozedur mit Variablenparametern genauso zu gestalten wie bei der Prozedur mit Werteparametern:

(1) Die Anzahl der aktuellen und formalen Parameter muß übereinstimmen.

(2) Die einander zugeordneten aktuellen und formalen Parameter müssen denselben Datentyp haben. Für **String-Variablen** wird in Turbo-Pascal jedoch eine Ausnahme zugelassen, die am Ende des Kapitels im Abschnitt über besondere Compiler-Optionen behandelt wird.

Auch für die Angabe der Variablen in der formalen Parameterliste gelten bei der Prozedur mit Variablenparametern dieselben Regeln wie bei der Prozedur mit Werteparametern:

(1) Zu jeder Variablen ist ein Datentyp anzugeben.

Beispiel: (var A: integer);

(2) Variablen desselben Datentyps können in einer Auflistung zusammengefaßt werden, hinter der der Datentyp angegeben wird.

Beispiel: (var A,B,C: integer);

(3) Vereinbarungen von Variablen mit unterschiedlichem Datentyp sind durch Semikolon voneinander zu trennen.

Beispiel: (var A,B,C: integer; X,Y: real);

(4) Die Angabe des Datentyps muß aus einem einfachen Bezeichner bestehen.

Ablaufsteuerung

Bei Starten des Programms werden auf Grund der Variablenvereinbarung

```
var Arbeitszeit, Fahrzeit, SummeArbeitszeit, SummeFahrzeit:
real;
```

Speicherplätze im Arbeitsspeicher unter den Namen Arbeitszeit, Fahrzeit usw. eingerichtet. Mit dem Prozeduraufruf

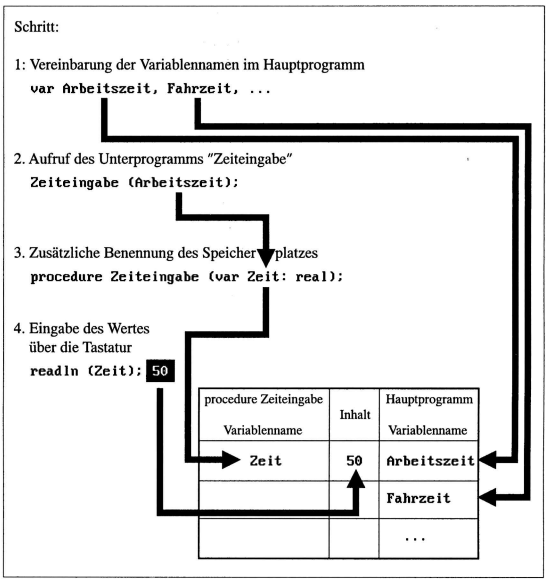
```
                    Zeiteingabe (Arbeitszeit)
in Verbindung mit        ↑
der Prozedurverein-      |
barung                   ↓
        procedure Zeiteingabe (var Zeit: real);
```

erhält der Speicherplatz für Arbeitszeit zusätzlich den Namen des Variablenparameters Zeit. Zur Laufzeit der Prozedur hat der Speicherplatz, der im Hauptprogramm mit Arbeitszeit angesprochen wird, den Namen Zeit.

Die bei der Anweisung (Zeile 7)

```
readln (Zeit)
```

eingegebene Zahl wird exakt in den Speicherplatz geschrieben, der zur Laufzeit des Hauptprogramms mit dem Namen Arbeitszeit adressiert ist.



Entsprechendes gilt für die Variable Fahrzeit. Beim Aufruf der Prozedur mittels

Zeiteingabe (Fahrzeit)

in Verbindung mit der Prozedurvereinbarung

procedure Zeiteingabe (var Zeit: real);

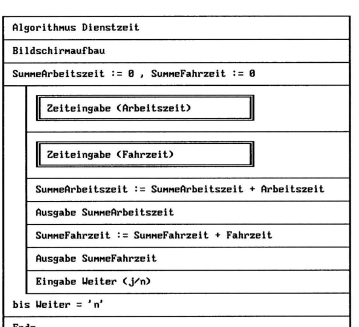
erhält der Speicherplatz für Fahrzeit den Namen Zeit. Bei der Ausführung der Anweisung in Zeile 7

```
readln (Zeit)
```

wird die eingegebene Zahl auf dem Speicherplatz abgelegt, der im Hauptprogramm unter dem Namen Fahrzeit geführt wird.
In der englischsprachigen Literatur wird die Prozedur mit den variablen Parametern mit «call by reference» charakterisiert. Darin kommt zum Ausdruck, daß beispielsweise beim Aufruf der Eingabe-Prozedur mittels

```
Zeiteingabe (Arbeitszeit)
```

der Prozedur Zeiteingabe mitgeteilt wird, daß der formale Parameter Zeit dieselbe Speicheradresse erhält wie der aktuelle Parameter Arbeitszeit. Im Vergleich zur Prozedur mit Werteparametern wird also bei der Prozedur mit Variablenparametern kein neuer Speicherplatz für den formalen Parameter eingerichtet. Vielmehr werden nur Namen (oder genauer: Speicheradressen) ausgetauscht.
Für die Entwicklung sowie die Dokumentation eines Programms ist es wichtig, daß die Anwendung von Prozeduren auch im Struktogramm kenntlich gemacht wird. Das obige Struktogramm zu «Dienstzeit» (Seite 233) ist dann wie folgt zu gestalten:



```
Prozedur Zeiteingabe (Variablenparameter: Zeit)
  Eingabe Zeit
bis Zeit >= 0 und Zeit <= 10
Ende
```

Modularisierte Programmierung

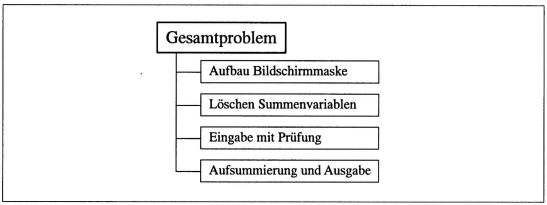
In Pascal können Unterprogramme in Form einer
(a) einfachen, parameterlosen Prozedur
(b) Prozedur mit Parametern
- Werteparametern
- Variablenparametern

gestaltet werden. Durch die vielfältigen Ausgestaltungsmöglichkeiten von Prozeduren erleichtert Pascal den Entwurf von Programmen. Prozeduren unterstützen nämlich eine Modularisierung von Programmen. Modularisieren heißt, ein Gesamtproblem in Teilprobleme (Module) aufzuteilen, die sich relativ einfach programmieren lassen. Aus der Summe der Teillösungen ergibt sich dann die Lösung des Gesamtproblems.

```
program

procedure A

procedure B

procedure C

procedure D

begin     (* Hauptprogramm *)
  A       (* Prozeduraufruf A *)
  B       (* Prozeduraufruf B *)
  C       (* Prozeduraufruf C *)
  D       (* Prozeduraufruf D *)
end.
```

Das Modulkonzept soll am Beispiel des Problems Dienstzeit dargelegt werden. Das Gesamtproblem läßt sich in Teilprobleme zergliedern:
(a) Aufbau der Bildschirmmaske

(b) Löschen der Summenvariablen (SummeArbeitszeit, SummeFahrzeit)
(c) Eingabe der Stunden für Arbeitszeit und Fahrzeit mit Prüfung auf Plausibilität
(d) Aufsummieren der Arbeitszeit und Fahrzeit mit Anzeige von SummeArbeitszeit und SummeFahrzeit



Faßt man die Anweisungen, die die Teilprobleme lösen, in Prozeduren zusammen, so ergibt sich folgendes Programm:

```
 0  program Dienstzeit_3;
    uses crt;                                          (* ab Version 4 *)
 1  var Arbeitszeit, Fahrzeit, SummeArbeitszeit, SummeFahrzeit: real;
 2      Weiter: char;

 3    procedure Zeiteingabe (var Zeit: real; Spalte, Zeile: integer);
 4    begin
 5      repeat
 6        gotoxy (Spalte,Zeile); ClrEol;
 7        readln (Zeit);
 8      until (Zeit >= 0) and (Zeit <= 10);
 9    end;

10    procedure Bildschirmmaske;
11    begin
12      ClrScr;
13      gotoxy ( 5,10); write ('Arbeitszeit ..: ');
14      gotoxy ( 5,12); write ('Fahrzeit .....: ');
15      gotoxy (10,14); write ('Weiter (j/n) : ');
16      gotoxy ( 5,17); write ('Summe Arbeitszeit:');
17      gotoxy ( 5,19); write ('Summe Fahrzeit:   ');
18    end;

19    procedure Aufsummierung_und_Anzeige (var Summe: real; Zeit: real;
                                           Spalte, Zeile: integer);
20    begin
21      Summe := Summe + Zeit;
22      gotoxy (Spalte,Zeile); write (Summe:8:2);
23    end;

24  begin   (* Hauptprogramm *)
25    Bildschirmmaske;
26    SummeFahrzeit := 0; SummeArbeitszeit := 0;
27    repeat
28      Zeiteingabe (Arbeitszeit, 22,10);
29      Zeiteingabe (Fahrzeit, 22,12);
30      Aufsummierung_und_Anzeige (SummeArbeitszeit, Arbeitszeit,23,17);
31      Aufsummierung_und_Anzeige (SummeFahrzeit, Fahrzeit,23,19);
32      gotoxy (25,14); read (Weiter);
33    until Weiter = 'n';
34  end.
```

Das Hauptprogramm besteht jetzt fast nur noch aus Prozeduraufrufen und Anweisungen zur Steuerung des Gesamtprogramms.

Mischformen bei Parametern

In dem Programm Dienstzeit_2 (Seite 235) wurde das Problem der Cursorsteuerung zunächst ausgeklammert. Hier wird es nun wieder aufgegriffen werden.

Wie Sie aus der Prozedurvereinbarung in Zeile 3

```
procedure Zeiteingabe (var Zeit: real;Spalte,Zeile: integer);
```

erkennen können, besteht die Liste der formalen Parameter aus einem Variablenparameter und aus Werteparametern. Zeit wurde als Variablenparameter, Spalte und Zeile wurden als Werteparameter vereinbart. Beide Formen von Parametern sind also innerhalb einer Prozedur zulässig.
Wenn Sie den Prozeduraufruf in Zeile 28

```
                                    Zeiteingabe (Arbeitszeit, 22,10);
mit dem Prozedurkopf
der Prozedur Zeiteingabe
procedure Zeiteingabe (var Zeit: real; Spalte, Zeile: integer);
```

in Beziehung setzen, so erkennen Sie, daß der Variablenparameter Zeit dem Speicherplatz von Arbeitszeit zugeordnet ist. Den Werteparametern Spalte und Zeile werden die ganzzahligen Konstanten 22 und 10 zugewiesen:

```
Spalte  := 22
Zeile   := 10
```

Somit stehen die Angaben, die für die Anweisung gotoxy (Spalte, Zeile) in der Programmzeile 6 benötigt werden, in den lokalen Variablen Spalte und Zeile zur Verfügung.
Die verschiedenen Möglichkeiten, Prozeduren auszugestalten, erleichtern es, modular aufgebaute Problemlösungen programmtechnisch umzusetzen. Die Methode der modularisierten Programmierung ist bei umfangreicheren Problemstellungen unbedingt zu empfehlen. Hier besteht nämlich oft die Gefahr, vor lauter Einzelproblemen den Überblick zu verlieren. Deshalb sollten Teillösungen in Prozeduren ausgelagert werden.
Im Hauptprogramm werden die Prozeduren verkettet, so daß sich daraus die Lösung des Gesamtproblems ergibt.

Besondere Compiler-Option

Es besteht die strenge Vorschrift, daß die Datentypen der formalen und aktuellen Parameter exakt übereinstimmen müssen. Das kann, wenn String-Variablen als Parameter verwendet werden, zu einer

umständlichen Programmierung führen. Für den Turbo-Pascal-Compiler sind zum Beispiel string[80] und string[50] unterschiedliche Datentypen, weil sie verschiedene Längenangaben haben.

Um die Gestaltung von Prozeduren mit Variablenparametern, in denen Zeichenketten bearbeitet werden, zu vereinfachen, besitzt Turbo-Pascal einen Compilerbefehl, der die strenge Typenprüfung aufhebt. Das sei an Hand einer Erweiterung des Programms Bezeichner_Test dargelegt.

In dem Programm Bezeichner_Test werden zwei Zeichenketten von unterschiedlicher Länge eingegeben. In die String-Variable Eingabetext können bis zu 80 Zeichen, in die String-Variable KurzerText bis zu 10 Zeichen eingegeben werden. Beide Zeichenketten sollen mit der Prozedur Ausgabe am Bildschirm angezeigt werden.

```
program Bezeichner_Test;
type string80  = string [80];
var Eingabetext: string [80];
    KurzerText : string [10];

procedure Ausgabe (var Ausgabetext: string80);
begin
  writeln (Ausgabetext);
end;

begin                                   (* Hauptprogramm *);
  readln (Eingabetext);
  Ausgabe (Eingabetext);
  readln (KurzerText);
  Ausgabe (KurzerText);
end.
```

Rufen Sie das vorher abgespeicherte Programm Bezeichner_Test wieder auf, und fügen Sie die hervorgehobenen Erweiterungen in das Programm ein!

Wenn Sie dieses Programm übersetzen lassen, meldet das System den Fehler:

```
Fehler 26: Typen nicht miteinander vereinbar.
```

Der Fehler besagt, daß der Compiler bei der Überprüfung der Datentypen festgestellt hat, daß sie nicht übereinstimmen. Sollen die String-Variablen, die mit unterschiedlicher Länge vereinbart worden

sind, aber trotzdem über die Prozedur Ausgabe verarbeitet werden, so muß die Typprüfung von String-Variablen abgeschaltet werden. Das bewirken Sie mit dem Compilerschalter

(*$V–*) bzw. {$V–}

der an den Anfang des Programms gesetzt wird. Der Programmanfang lautet dann:

```
program Bezeichner_Test;
(*$V-*)
type string80  = string [80];
var Eingabetext: string [80];
    KurzerText : string [10];
...
```

Forward-Referenz

Wird aus einer Prozedur eine weitere Prozedur aufgerufen, so sollte die aufgerufene Prozedur vorher vereinbart worden sein. Nun kann es aber die Problemsituation erfordern, daß diese Regel nicht eingehalten werden kann. Das ist beispielsweise immer dann der Fall, wenn zwei Prozeduren sich gegenseitig aufrufen.

Beispiel:

```
program beispiel;
var (* Vereinbarungsteil *)
procedure Eins (Liste der formalen Parameter); forward
procedure Zwei (Liste der formalen Parameter);
begin
   Eins (Liste der aktuellen Parameter); (* Aufruf der Prozedur
                                            Eins *)
end;
procedure Eins;
begin
   Zwei(Liste der aktuellen Parameter); (* Aufruf der Prozedur
                                           Zwei *)
end;
   ...
```

In diesem Beispiel ruft die Prozedur Eins die Prozedur Zwei auf. Aus der Prozedur Zwei erfolgt ein Aufruf der Prozedur Eins. Die Vor-

schrift, daß eine Prozedur vor ihrem Aufruf vereinbart worden sein muß, kann also nicht eingehalten werden.
In einem solchen Fall wird der Kopf der Prozedur, deren Anweisungsteil später dargelegt wird, vorweg mit dem Zusatz **forward** deklariert. Bei dem später dargestellten Anweisungsteil der vorweg vereinbarten Prozedur wird nur noch ein verkürzter Prozedurkopf in der Form

```
procedure Prozedurname;
```

angegeben. Formale Parameter werden nicht mehr dahinter vermerkt. Sie sind ja bereits bei der «Vorweg-Deklaration» festgelegt worden.

7.4.1 Zusammenfassung

- Prozeduren mit Werteparametern ermöglichen es, Folgen von Anweisungen, die sich nur darin unterscheiden, daß sie mit unterschiedlichen Variablen arbeiten, zu einer Prozedur zusammenzufassen.
- Im Prozeduraufruf müssen Variablen als aktuelle Parameter angegeben werden.
- Variablenparameter werden durch ein vorgestelltes **var** gekennzeichnet.
- In einer Prozedurvereinbarung können sowohl Werteparameter als auch Variablenparameter vereinbart werden.
- Die Prozedurtechnik unterstützt die modularisierte Programmierung.
- Bei der Übergabe von Zeichenkettenvariablen kann die strenge Datentypüberprüfung mittels des Compilerbefehls (*$V-*) abgeschaltet werden.
- Durch die forward-Referenz wird der Reihenfolgekonflikt bei der Anordnung von Prozeduren vermieden.

7.4.2 Übungen

1. Aufgabe
Welcher Unterschied besteht zwischen einem Werteparameter und einem Variablenparameter?

2. Aufgabe
Schreiben Sie eine Prozedur, mit der im Programm Dienstzeit_3 die beiden Variablen SummeArbeitszeit und SummeFahrzeit gelöscht werden.

3. Aufgabe
Gestalten Sie in dem Programm GasverbrauchSumme, das als Lösung der Aufgabe 2 zu Übung 7.2.2 angegeben wurde, die Prozedur Zaehlereingabe in der Weise um, daß sie durch

Zaehlereingabe (AltZaehler,37,6)
sowie
Zaehlereingabe (NeuZaehler,37,8)
aufgerufen werden kann.

4. Aufgabe
Entwickeln Sie ein Programm, das drei Zahlen, die in beliebiger Reihenfolge eingegeben werden, in aufsteigend sortierter Reihenfolge ausgibt.
Lösungshinweis:
Bei diesem Programm sind die Inhalte von Variablen zu vertauschen. Wenn beispielsweise in die Variable A der Wert 17 und in Variable B der Wert 13 eingegeben wurde, sind wegen der aufsteigenden Sortierfolge die Werte umzuspeichern. Für die Umspeicherung ist eine Hilfsvariable heranzuziehen, über die die Tauschoperation vorgenommen wird:

```
if A > B
    then begin
    Hilfsspeicher := A;
    A := B;
    B := Hilfsspeicher;
    end;
```

Diese algorithmische Konstruktion nennt man auch Dreieckstausch.
Um sich die erforderlichen Vergleichsanweisungen und Tauschoperationen klarzumachen, sollten Sie folgendes Zahlenbeispiel manuell durchspielen.

	17	13	11
Wenn A > B	A	B	C
Dann tausche A und B			
	13	17	11
Zustand nach Tausch:	A	B	C
Wenn B > C			
Dann tausche B und C	13	11	17
Zustand nach Tausch:	A	B	C
Wenn...			
Dann tausche...			
Zustand nach Tausch:	A	B	C

Gestalten Sie das Programm in der Weise, daß die Tauschoperation in einer Prozedur mit Variablenübergabe erfolgt.

7.4.3 Lösungen

1. Aufgabe
Für einen Werteparameter wird ein neuer Speicherplatz angelegt. Beim Variablenparameter wird die übergebene Variable unter dem Namen des formalen Variablenparameters angesprochen.

2. Aufgabe

```
procedure Loeschen (var Variable: real);
begin
  Variable := 0;
end;
```

3. Aufgabe

```
procedure Zaehlereingabe (var Zaehlerstand : real;
                              Spalte, Zeile: integer);

  procedure Fehlermeldung;
  var Bestaetigung: char;
  begin
    gotoxy (5,20);
    write ('Eingabe unzulässig! <Return> drücken');
    gotoxy (5,20); readln (Bestaetigung);
    gotoxy (5,20); DelLine;
  end;

begin                              (* Anweisungsteil von Zaehlereingabe *)
  repeat
    gotoxy (Spalte,Zeile); ClrEol; readln (Zaehlerstand);
    if (Zaehlerstand < 0) or (Zaehlerstand > 100000.0)
      then Fehlermeldung;
  until (Zaehlerstand >= 0) and (Zaehlerstand < 100000.0);
end;
```

4. Aufgabe

Das Programm lautet:

```
program ZahlenSortierung;
uses crt;                                          (* ab Version 4 *)
var A,B,C: real;

  procedure Tauschen (var X,Y: real);
  var Zwischenwert: real;
  begin
    Zwischenwert := X;
    X := Y;
    Y := Zwischenwert;
  end;

begin (* Hauptprogramm *)
  ClrScr;
  writeln ('Eingabe von 3 Zahlen');
  write ('a = '); readln (A);
  write ('b = '); readln (B);
  write ('c = '); readln (C);
  if A > B
    then Tauschen (A,B);
  if B > C
    then Tauschen (B,C);
  if A > B
    then Tauschen (A,B);
  writeln ('a = ',A:5:3,' b = ',B:5:2,' c = ',C:5:2);
end.
```

7.5 Funktionen

Eine Funktion ähnelt in vieler Hinsicht einer Prozedur. Im Gegensatz zur Prozedur liefert eine Funktion stets genau einen einfachen Wert an das aufrufende Programm. Das sei am Beispiel der kaufmännischen Rundung dargelegt.

***Programmbeispiel* Bareinkaufspreis**
Ein Einzelhandelsgeschäft berechnet beim Einkauf von Waren den Bareinkaufspreis nach folgendem Kalkulationsschema:

Listeneinkaufspreis	
– Liefererrabatt	(in % vom Listeneinkaufspreis)
Zieleinkaufspreis	
– Liefererskonto	(in % von Zieleinkaufspreis)
Bareinkaufspreis	

Beispielrechnung für eine Ware, deren Listenpreis 200,00 DM beträgt, wenn der Lieferer 10 % Rabatt und 2 % Skonto gewährt:

Listeneinkaufspreis	200,00
– 10 % Liefererrabatt	20,00
Zieleinkaufspreis	180,00
– 2 % Liefererskonto	3,60
Bareinkaufspreis	176,40

Der Einzelhändler muß also 176,40 DM für die Ware bezahlen.
Zur Berechnung des Bareinkaufspreises sind die Werte für

Listenpreis
Rabattsatz
Skontosatz

einzugeben. Die errechneten Beträge für Liefererrabatt und Liefererskonto sind nach kaufmännischer Rundungsweise zu runden:

	gerundet
34,565 —>	34,57
34,564 —>	34,56

Die dritte Nachkommastelle ist also entscheidend.
Die Ausgabe der Daten soll entsprechend der obigen Beispielrechnung erfolgen. Da aus der Beispielrechnung auch der Ablauf der Arbeitsschritte zu ersehen ist, wird auf die Entwicklung des Algorithmus hier verzichtet.
Mit folgendem Programm wird der Bareinkaufspreis errechnet.

```
0  program Berechnung_Bareinkaufspreis;
   uses crt;                                        (* ab Version 4 *)
1  var Listenpreis, Rabatt, Skonto, Zieleinkaufspreis,
       Bareinkaufspreis,
2      Rabattsatz, Skontosatz: real;

3    function Runden (Wert: real): real;
4    var Zwischenwert : real;
5    begin
6      Zwischenwert := int(Wert*100 + 0.500001);
7      Runden := Zwischenwert / 100;
8    end;

9  begin
10   ClrScr;                                        (* Eingabeteil *)
11   write ('Listenpreis: '); readln (Listenpreis);
12   write ('Rabattsatz:  '); readln (Rabattsatz);
13   write ('Skontosatz:  '); readln (Skontosatz);
                                                    (* Berechnungsteil *)
14   Rabatt := Runden (Listenpreis * Rabattsatz / 100);
15   Zieleinkaufspreis := Listenpreis - Rabatt;
16   Skonto := Runden (Zieleinkaufspreis * Skontosatz / 100);

17   Bareinkaufspreis := Zieleinkaufspreis - Skonto;
                                                    (* Ausgabeteil *)
18   writeln; writeln;
19   writeln ('Listenpreis       ',Listenpreis:10:2);
20   writeln ('Rabatt            ',Rabatt:10:2);
21   writeln ('--------------------------');
22   writeln ('Zieleinkaufspreis ',Zieleinkaufspreis:10:2);
23   writeln ('Skonto            ',Skonto:10:2);
24   writeln ('--------------------------');
25   writeln ('Bareinkaufspreis  ',Bareinkaufspreis:10:2);
26 end.
```

Die Funktion Runden ist in den Zeilen 3 bis 8 dargestellt. Sie setzt sich zusammen aus dem Funktionskopf (Zeile 3), dem Vereinbarungsteil für lokale Variablen (Zeile 4) und dem Anweisungsteil der Funktion, der sich über die Zeilen 5 bis 8 erstreckt. Der prinzipielle Aufbau gleicht also dem einer Prozedur.

Vereinbarung und Aufruf

Die Vereinbarung einer Funktion wird im Funktionskopf vorgenommen. Sie wird eingeleitet mit dem Schlüsselwort **function**. Dahinter folgt der Name der Funktion. Daran schließt sich in runden Klammern die Angabe der formalen Parameter an. Darauf folgt ein Doppelpunkt, hinter dem der Datentyp des Wertes, den die Funktion liefern soll, vermerkt wird.
Allgemein lautet die Vereinbarung einer Funktion:

```
function Funktionsname (Liste der formalen Parameter): Datentyp;
```

In der Liste der formalen Parameter können sowohl Werte- als auch Variablenparameter verwendet werden. Innerhalb der Funktion muß das Ergebnis, das an die aufrufende Stelle geliefert werden soll, auf den Funktionsnamen zugewiesen werden:

```
Funktionsname := Ausdruck
```

Die Funktion wird in einem Ausdruck aufgerufen über ihren Namen. Hinter dem Namen ist in runden Klammern die Liste der aktuellen Parameter anzugeben.

```
Funktionsname (Liste der aktuellen Parameter)
```

Wie bei Prozeduren müssen auch bei Funktionen die formalen und aktuellen Parameter hinsichtlich ihrer Anzahl und ihres Datentyps übereinstimmen.

Ablaufsteuerung

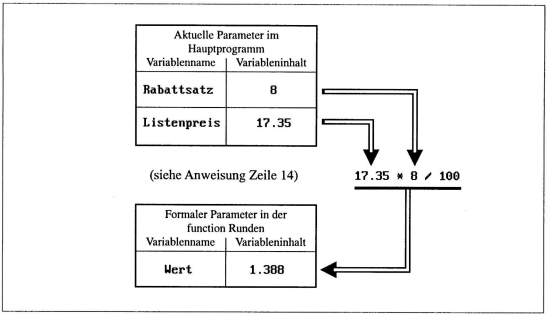
Die Funktion Runden wird zum erstenmal in der Anweisung (Zeile 14)

```
Rabatt := Runden (Listenpreis * Rabattsatz / 100);
```

angesprochen. Dabei wird der Wert des Ausdrucks

```
Listenpreis * Rabattsatz / 100
```

der Variablen Wert, die im Prozedurkopf als formaler Parameter angegeben ist, zugewiesen.



In der Funktion wird die lokale Variable Zwischenwert angelegt, in der durch die Anweisung

```
Zwischenwert := int (Wert*100 + 0.500001);
```

ein Rundungszwischenergebnis gespeichert wird. Daß hier ein Wert von 0.500001 statt 0.5 hinzuaddiert wird, liegt in der Art der internen Zahlendarstellung begründet. Die Zahl 1.5 beispielsweise wird intern mit 1.499999999999999 dargestellt. Würde nur ein Wert von 0.5 addiert, so käme es nicht zu der gewünschten Aufrundung. Bezogen auf das Zahlenbeispiel ergeben sich folgende Rechenoperationen:

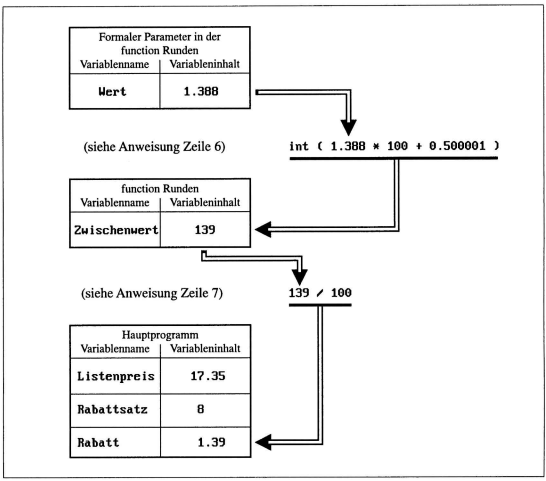
```
1.388 * 100 + 0.500001 = 138.8 + 0.500001 = 139.300001
```

Durch die Standardfunktion **int** werden die Nachkommastellen abgeschnitten, so daß sich in der Variablen Zwischenwert der Wert

```
int (139.300001) = 139
```

bildet.
Die richtig gerundete Zahl ergibt sich, indem der Inhalt von Zwischenwert durch 100 dividiert wird. Dieser Wert stellt das gewünschte Ergebnis dar (Ergebnisparameter). Deshalb wird er nicht irgendeiner Variablen, sondern direkt dem Funktionsnamen zugewiesen:

Runden := Zwischenwert / 100;



Nach Abschluß der Funktionsbearbeitung steht die gerundete Zahl in der Variablen Rabatt im Hauptprogramm zur Verfügung.

7.5.1 Zusammenfassung

- Eine Funktion liefert stets einen einfachen Wert an die aufrufende Stelle.
- In der Funktionsvereinbarung ist der Datentyp des Wertes anzugeben, den die Funktion liefert.
- Hinsichtlich der formalen und aktuellen Parameter gelten dieselben Vorschriften wie bei der Prozedur.
- Eine Funktion wird in einem Ausdruck durch Angabe des Funktionsnamens aufgerufen.

7.5.2 Übungen

1. Aufgabe
Ergänzen Sie folgende Funktionsvereinbarung:

```
function FTest (x,y: real)
```

2. Aufgabe
Die Funktion Runden, die in dem Programm Berechnung_Bareinkaufspreis mit

```
function Runden (Wert: real): real;
var Zwischenwert : real;
begin
  Zwischenwert := int(Wert*100 + 0.500001);
  Runden := Zwischenwert / 100;
end;
```

angegeben wurde, kann noch um eine Anweisung gekürzt werden. Wie ist die Funktion dann zu formulieren?

3. Aufgabe
Entwickeln Sie ein Programm, mit dem die Potenzwerte für

x^1 x^2 x^3 x^4 x^5 x^6

berechnet und ausgedruckt werden. x ist eine ganze Zahl, die die Werte von 2 bis 8 annehmen soll.
Zahlenbeispiel:

2	4	8	16	32	64
3	9	27	81	243	729
...					
8	64	512	4096	32768	262144

Lösungshinweise:
Der Potenzwert kann, wenn die Hochzahl eine positive Ganzzahl ist, durch fortgesetzte Multiplikation errechnet werden:
$2^3=2*2*2$.

Das Struktogramm hat die Form:

```
Algorithmus Potenzreihe
Bildschirm löschen
Für x := 2 bis 8
    Für y := 1 bis 6
        Potenzuert := Potenz (x,y)
        Ausgabe Potenzuert
    Zeilenschaltung
Ende
```

```
Funktion Potenz  (Werteparameter i,n)
Zwischenuert := 1
Für Zaehler := 1 bis n
    Zwischenuert := Zwischenuert * i
Potenz := Zwischenuert
Ende
```

4. Aufgabe

Für die Potenzrechnung gibt es in Turbo-Pascal nur die beiden Standardfunktionen für das Quadrat (sqr) und die Quadratwurzel (sqrt) einer Zahl. Möchten Sie andere Potenzaufgaben lösen, so müssen Sie sich mit Hilfe der Logarithmusrechnung Ihre eigene Potenzfunktion programmieren. (Wenn Sie nicht mehr wissen, wie das geht, schauen Sie bitte in einem Mathematikbuch nach.)
Erstellen Sie ein Programm, mit dem für die Zahlen von 2 bis 10 der Wert der 3. Wurzel sowie der 3. Potenz berechnet und angezeigt wird.

Ausgabe:

```
 2   1.2599     8
 3   1.4422    27
     ...
10   2.1544 1000
```

Lösungshinweise:
In Pascal werden dazu die Standardfunktionen ln (= logarithmus naturalis = Logarithmus zur Basis e) und die Exponentialfunktion exp benötigt, die den Potenzwert von e^x berechnet.
Der Potenzwert von 3^4 bestimmt sich unter Anwendung der beiden Standardfunktionen ln und exp aus:

```
Potenzwert := exp(ln(3)*4)
```

Die 4. Wurzel aus 3 wird bestimmt durch

```
Potenzwert := exp(ln(3)*1/4)
```

Der Potenzwert von x^y errechnet sich also aus

```
Potenzwert := exp (ln (x) * y)
```

7.5.3 Lösungen

1. Aufgabe
Es fehlt die Angabe, welchen Datentyp das Ergebnis der Funktion hat. Die vollständige Vereinbarung könnte beispielsweise lauten:

```
function FTest (x, y: real): real;
```

2. Aufgabe

```
function Runden (Wert: real): real;
begin
  Runden := int(Wert*100 + 0.500001) / 100;
end;
```

3. Aufgabe

```
program Potenzreihe;
uses crt;                                          (* ab Version 4 *)
var x,y : integer;
    Potenzwert : real;

  function Potenz (i,n: integer): real;
  var Zaehler : integer;
      Zwischenwert : real;
  begin
    Zwischenwert := 1;
    for Zaehler := 1 to n do
      begin
        Zwischenwert := Zwischenwert * i;
        Potenz := Zwischenwert;
      end;
  end;

begin                          (* Hauptprogramm *)
  ClrScr;
  for x := 2 to 8 do
    begin
      for y := 1 to 6 do
        begin
          Potenzwert := Potenz (x,y);
          write (Potenzwert:8:0);
        end;
      writeln;
    end;
end.
```

4. Aufgabe

```
program Dritte_Wurzel_und_Potenz;
var n: integer;
    Hoch3, Wurzel3: real;

  function Potenzwert (X: integer; Y:real): real;
  begin
    Potenzwert := exp (ln (X)*Y);
  end;

begin                                (* Hauptprogramm *)
  ClrScr;
  for n := 2 to 10 do
    begin
      Wurzel3 := Potenzwert (n,1/3);
      Hoch3 := Potenzwert (n,3);
      writeln (n:3, Wurzel3:10:4, Hoch3:6:0);
    end;
end.
```

7.6 Rekursion

Es war einmal ein Mann, der hatte 7 Kinder, und die Kinder sprachen: «Vater, erzähl uns eine Geschichte». Und der Vater begann: «Es war einmal ein Mann, der hatte 7 Kinder, und die Kinder sprachen: ‹Vater, erzähl uns eine Geschichte›. Und der Vater begann: ‹Es war einmal ein Mann, der hatte 7 Kinder, und die Kinder sprachen: ...›»
Diese «unendliche» Geschichte ist ein Beispiel für eine Rekursion, da sie sich immer wieder selber aufruft. In Pascal kann man Prozeduren oder Funktionen rekursiv, das heißt in Form einer Rekursion schreiben.

Beispiel:

```
procedure irgendwas;
begin
   Anweisung;
   irgendwas; {rekursiver Aufruf}
end;
```

Rekursiv nennt man also die Prozedur oder Funktion, die sich in ihrem Anweisungsteil selbst aufruft. In dem nachstehenden Programm wird die Abschreibungstabelle für lineare Abschreibung (siehe Kapitel 5) in einer rekursiven Prozedur erzeugt.

```
program Abschreibung_rekursiv;
uses crt;                                          (* ab Version 4 *)
var Nutzungsdauer: integer;
    Anschaffungswert, Abschreibung : real;

  procedure AbschreibTab ( Peranfwert, Abschr: real);
  var Perendwert: real;
  begin
    Perendwert := Peranfwert - Abschr;
    writeln (Peranfwert:10:2, Abschr:10:2, Perendwert:10:2);
    if Perendwert > 0
      then AbschreibTab (Perendwert, Abschr);
  end;

begin                              (* Hauptprogramm *);
  ClrScr;
  write ('Anschaffungswert: '); readln (Anschaffungswert);
  write ('Nutzungsdauer   : '); readln (Nutzungsdauer);
  Abschreibung := Anschaffungswert / Nutzungsdauer;
  AbschreibTab (Anschaffungswert, Abschreibung);
end.
```

Die Prozedur AbschreibTab wird von zwei verschiedenen Stellen aufgerufen. Der erste Aufruf erfolgt aus dem Hauptprogramm:

```
AbschreibTab (Anschaffungswert, Abschreibung);
```

Der zweite und die nachfolgenden Aufrufe der Prozedur werden aus der Prozedur selbst vorgenommen:

```
if Perendwert > 0 then AbschreibTab (Perendwert, Abschr);
```

Diese rekursive Form der Prozedurgestaltung kommt einer Schleifenkonstruktion gleich. Die Prozedur ruft sich selbst auf, wenn der Wert der lokalen Variablen Perendwert größer als null ist. Die Prozedur wird verlassen, wenn der Inhalt von Perendwert kleiner als oder gleich null ist.
Bei jedem erneuten Aufruf der Prozedur werden für die Werteparameter neue Speicherplätze angelegt. Deshalb kann es zu einem Programmabsturz kommen, wenn die rekursive Prozedur beziehungsweise Funktion zu häufig aufgerufen wird. Der Speicherplatz reicht dann nicht mehr aus, um die für die Programmsteuerung benötigten Daten speichern zu können. Der Programmlauf wird abgebrochen mit der Meldung: «Run-time error FF». Das bedeutet Stack-/Heap-Kollision (Zusammenbruch des Stapelspeichers), weil nicht genügend freier Speicherplatz vorhanden ist.

7.6.1 Zusammenfassung

- Eine Prozedur beziehungsweise Funktion, die sich selbst aufruft, nennt man rekursiv.
- Eine rekursive Prozedur beziehungsweise Funktion muß unbedingt eine Endebedingung enthalten, um eine unendliche Rekursion zu vermeiden.

7.6.2 Übungen

1. Aufgabe
Untersuchen Sie das Programm, und geben Sie die Entwicklung der Werte für x und y in der Wertebelegungstabelle an.

```
program RekursionsBeispiel;

procedure wertbildung (y:real);
var x:real;
begin
  x := y * 2;
  writeln (x:10:4);
  if x < 50
    then wertbildung (x);
end;

begin
  wertbildung(3);
end.
```

Wertebelegungstabelle

Aufruf	y	x
1	3	6

2. Aufgabe

Durch welche Anweisung kann eine Rekursion ersetzt werden?
(a) durch eine Auswahlanweisung
(b) durch eine Wiederholungsanweisung
(c) durch eine Sprunganweisung

7.6.3 Lösungen

1. Aufgabe

Wertebelegungstabelle

Aufruf	y	x
1	3	6
2	6	12
3	12	24
4	24	48
5	48	96

2. Aufgabe

Antwort b) ist richtig. Jede rekursive Prozedur läßt sich prinzipiell durch eine Schleifenanweisung ersetzen.

7.7 Standardprozeduren und -funktionen

Turbo-Pascal bietet eine Vielzahl vordefinierter Prozeduren und Funktionen. Sie werden über festgelegte Namen aufgerufen. Bei den meisten Standardprozeduren/Standardfunktionen gibt es strenge Vorschriften bezüglich der Datentypen.

7.7.1 Arithmetische Funktionen

Absolut-Funktion

Funktionsaufruf	Datentyp des Eingabewertes	Datentyp des Ausgabewertes	Bedeutung
abs (i)	integer	integer	bildet den absoluten Wert
abs (r)	real	real	(Betrag) einer Zahl

Beispiel:

```
x := abs (-3.4)
```

In x bildet sich der Wert 3.4.

Dezimalrest-Funktion

Funktionsaufruf	Datentyp des Eingabewertes	Datentyp des Ausgabewertes	Bedeutung
frac (i)	integer	real	Abtrennen der Nach-
frac (r)	real	real	kommastellen von den Vorkommastellen

Beispiel:

In einem Programm wird als Datumsangabe nur die Tages- und Monatsangabe benötigt. Für diesen Fall könnte man beispielsweise das Datum für den 13. November als Dezimalzahl in der Form 13.11 eingeben. Anschließend soll das Datum in seinen Tages- beziehungsweise Monatsbestandteil aufgeteilt werden.

```
program Datumsaufspaltung;
var Datum : real;
    Tag, Monat: real;
begin
  write ('Eingabe Datum xx.xx '); readln (Datum);
  Monat := frac (Datum);
  Tag   := Datum - Monat;
  Monat := Monat * 100;
  writeln ('Tag:',Tag:2:0, ' Monat:',Monat:2:0);
end.
```

Wertbelegungstabelle

Datum	Monat	Tag
13.11		
	0.11	13
	11	

Exponential-Funktion

Funktions-aufruf	Datentyp des Eingabewertes	Datentyp des Ausgabewertes	Bedeutung
exp (i)	integer	real	Berechnet den Potenz-
exp (r)	real	real	wert von e^x

Beispiel:

y := exp (1)

In y bildet sich der Wert 2.71828.

Ganzzahl-Funktion

Funktions-aufruf	Datentyp des Eingabewertes	Datentyp des Ausgabewertes	Bedeutung
int (i)	integer	real	Abschneiden der Nach-
int (r)	real	real	kommastellen

Beispiel:

y := int (3.34);

In y bildet sich der Wert 3 (als Real-Wert).

y := int (–3.34);

In y bildet sich der Wert –4 (als Real-Wert).

Logarithmus-Funktion (natürlicher Logarithmus)

Funktions-aufruf	Datentyp des Eingabewertes	Datentyp des Ausgabewertes	Bedeutung
ln (i)	integer	real	Bildet den Logarithmus
ln (r)	real	real	einer Zahl zur Basis e

Beispiel:

y := ln (10)

In y bildet sich der Wert 2.30258.

Die Funktion ln wird in Verbindung mit der Funktion exp benötigt, um Potenzwerte zu berechnen. Für das folgende Programm gilt die Voraussetzung, daß die Basis als positive Zahl eingegeben werden muß.

```
program Potenz;
var Basis, Hochzahl, Potenzwert: real;
begin
  write ('Basis: '); readln (Basis);
  write ('Hochzahl: '); readln (Hochzahl);
  Potenzwert := exp (hochzahl * ln(basis));
  writeln ('Potenzwert: ', Potenzwert:10:4);
end.
```

Dialog: Basis: 25
Hochzahl: 2
Potenzwert: 625

Quadrat-Funktion

Funktions-aufruf	Datentyp des Eingabewertes	Datentyp des Ausgabewertes	Bedeutung
sqr (i)	integer	real	Bildet das Quadrat einer
sqr (r)	real	real	Zahl

Beispiel:

y := sqr (25)

In y bildet sich der Wert 625.

Quadratwurzel-Funktion

Funktions-aufruf	Datentyp des Eingabewertes	Datentyp des Ausgabewertes	Bedeutung
sqrt (i)	integer	real	Zieht die Quadratwurzel
sqrt (r)	real	real	aus einer Zahl

Beispiel:

y := sqrt (625)

In y bildet sich der Wert 25.

Winkelfunktionen

Funktions-aufruf	Datentyp des Eingabewertes	Datentyp des Ausgabewertes	Bedeutung
arctan (i)	integer	real	Berechnet im Bogen-maß den Hauptwert
arctan (r)	real	real	von arctan
cos (i)	integer	real	Berechnet den Cosinus
cos (r)	integer	real	im Bogenmaß
sin (i)	integer	real	Berechnet den Sinus im
sin (r)	real	real	Bogenmaß

7.7.2 Ordinale Funktionen

Folgende Funktionen sind anwendbar bei ordinalem Datentyp, also bei abzählbaren Typen (Integer, Byte, Boolean, Char).
Die Menge der Buchstaben oder der Integer-Zahlen ist beispielsweise ordinal, weil man bei diesen Datentypen zu jedem Element genau den Vorgänger beziehungsweise Nachfolger angeben kann.

Funktions-aufruf	Datentyp des Eingabewertes	Datentyp des Ausgabewertes	Bedeutung
pred (z)	ordinal	ordinal	Ermittelt den Vorgänger
succ (z)	ordinal	ordinal	Ermittelt den Nachfolger
odd (i)	integer	boolean	Überprüft eine Ganzzahl auf gerade beziehungsweise ungerade
dec (z)	ordinal	ordinal	Vermindert den Inhalt der Variablen z um 1
dec (z, n)	ordinal n: longint	ordinal	Vermindert den Inhalt der Variablen z um den Wert n
inc (z)	ordinal	ordinal	Erhöht den Inhalt der Variablen z um 1
inc (z, n)	ordinal n: longint	ordinal	Erhöht den Inhalt der Variablen z um den Wert n

Beispiele:

```
Zeichen := pred ('C')
```
In der Char-Variablen Zeichen bildet sich der Buchstabe B.

```
Zeichen := succ ('K')
```
In der Char-Variablen Zeichen bildet sich der Buchstabe L.

```
Ungerade := odd (7)
```
In der Boolean-Variable Ungerade ergibt sich der Wert true.

```
Ungerade := odd (8)
```
In der Boolean-Variable Ungerade ergibt sich der Wert false.

7.7.3 Umwandlungsfunktionen

Mit Hilfe von Umwandlungsfunktionen werden Werte von Variablen, die skalar oder auch nicht skalar sind, Variablen skalaren Typs zugewiesen.

Funktions-aufruf	Datentyp des Eingabewertes	Datentyp des Ausgabewertes	Bedeutung
chr (i)	integer	char	Umwandlung von Integer nach Char
ord (z)	ordinal	integer	Ermittelt den ordinalen Wert eines skalaren Datums

Beispiele:

Z := chr (65)

In der Char-Variablen Z bildet sich der Buchstabe A. Nach dem ASCII-Zeichensatz steht das A an 65. Stelle.

I := ord ('A')

Der Integer-Variablen I wird der Wert 65 zugewiesen, weil der Buchstabe A an 65. Stelle im ASCII-Zeichensatz steht.

Mit folgendem Programm können Sie sich den Zeichensatz Ihres Computers auf dem Bildschirm anzeigen lassen:

```
program Umwandlung;
var I : integer;
    Z : char;

begin
for I := 1 to 255 do
  begin
    Z := chr (I);
    write (Z);
  end;
end.
```

Das folgende Programm zeigt die den Großbuchstaben zugeordneten ordinalen Werte an:

```
program Ordinalfunktion;
var N, I : integer;
    Zeich:char;
begin
  Zeich := 'A';
  while Zeich <= 'Z' do
    begin
      I := ord (Zeich);
      writeln (Zeich,' = ',I:4);
      Zeich := succ (Zeich);
    end;
end.
```

Funktions-aufruf	Datentyp des Eingabewertes	Datentyp des Ausgabewertes	Bedeutung
round (r)	real	integer	Rundungsfunktion
trunc (r)	real	integer	Ermittelt den ganzzahligen Bestandteil einer Real-Zahl

Beispiele:

N := round (6.4); M := round (6.5)

In der Integer-Variablen N bildet sich der Wert 6. Der Integer-Variablen M wird der Wert 7 zugewiesen.

N := trunc (6.4)

In der Integer-Variablen N bildet sich der Wert 6.

7.7.4 Prozeduren und Funktionen für die Bildschirmsteuerung

Bildschirmprozeduren

Prozeduraufruf	Wirkung
ClrEol	Löscht alle Zeichen ab Cursorposition bis Zeilenende
ClrScr	Löscht den gesamten Bildschirm
CrtExit	Schickt den Terminal-Reset-String
CrtInit	Schickt den Terminal-Initialisierungsstring
DelLine	Löscht die Zeile, in der der Cursor steht. Die nachfolgenden Zeilen werden um eine nach oben verschoben.

Prozeduraufruf	Wirkung
Gotoxy (s, z)	Positioniert den Cursor auf die Spalte s, Zeile z. s und z sind vom Typ Integer.
InsLine	Fügt an der Cursorposition eine Leerzeile ein. Die nachfolgenden Zeilen werden um eine nach unten verschoben.
LowVideo	Die Zeichen werden in verminderter Helligkeit dargestellt.
NormVideo	Die Zeichen werden in normaler Helligkeit dargestellt.

Funktionen für Cursorabfrage

Funktionsaufruf	Bedeutung
Wherex	Ermittelt die Nummer der Spalte, in der sich der Cursor befindet
Wherey	Ermittelt die Nummer der Zeile, in der sich der Cursor befindet

7.7.5 Prozeduren und Funktionen zur Stringverarbeitung

Stringprozeduren

Prozeduraufruf	Wirkung
delete (Zeichvar, Stelle, Anzahl)	Löscht aus der Stringvariablen Zeichvar ab einer bestimmten Stelle eine bestimmte Anzahl von Zeichen. Die Angaben für Stelle und Anzahl müssen vom Typ Integer sein. Beispiel: Zeichvar := 'Düsseldorf'; delete (Zeichvar,2,3); Zeichvar hat jetzt den Inhalt Deldorf.
insert (Zkette, Zeichvar, Stelle)	Fügt in die Stringvariable Zeichvar ab einer bestimmten Stelle die Zeichenkette Zkette ein.

Prozeduraufruf	Wirkung
	Beispiel: Zeichvar := 'Delhorst'; insert ('men', Zeichvar,4); Zeichvar enthält jetzt den Wert Delmenhorst.
str (Zahlenwert, Zeichvar)	Wandelt eine Zahl in einen String um und speichert ihn in Zeichvar. Zahlenwert kann Integer oder Real sein. Beispiel: x := 1993; str (x:6,Zeichvar) Zeichvar enthält nun '1993'.
val (Zeichvar, Zahlvar, Code)	Wandelt in Abhängigkeit vom Datentyp von Zahlvar eine Zeichenkette in einen Integer- oder Real-Wert um. Die Integervariable Code weist den Wert 0 auf, wenn die Umwandlung korrekt verlaufen ist. Beispiele: Zeichvar := '1993'; val (Zeichvar, Jahr, Code); Die numerische Variable Jahr weist nun den numerischen Wert 1993 auf. Zeichvar := '12.05.1993'; val (Zeichvar, Datum, Code); Code weist den Wert 6 auf, da an der 6. Stelle der Zeichenkette ein zweiter Dezimalpunkt angetroffen wurde. Real-Werte dürfen nämlich nur einen Dezimalpunkt aufweisen. Der Inhalt von Datum ist nicht definiert.

Stringfunktionen

Prozeduraufruf	Wirkung
copy (Zeichvar, Stelle, Anzahl)	Kopiert aus einer Zeichenkette ab einer bestimmten Stelle eine bestimmte Anzahl von Zeichen. Beispiel: Zeichvar := 'Düsseldorf'; Zkette := copy (Zeichvar,2,5) Zkette enthält nun den String 'üssel'.
concat (Zkette1, Zkette2, ...)	Verknüpft Zeichenketten miteinander. Beispiel: Zk1 := 'üssel'; Zk2 := 'D'; Zk3 := 'dorf'; Geskette := concat (Zk2, Zk1, Zk3) Geskette enthält nun 'Düsseldorf'. Anmerkung: Das gleiche Ergebnis erzielt man mit Geskette := Zk2 + Zk1 + Zk3
length (Zeichvar)	Ermittelt die Länge eines Stringausdrucks. Beispiel: In die Variable Ort wurde der Name Düsseldorf eingegeben. Mittels length wird die Länge des eingegebenen Wortes festgestellt. Ortlaenge := length (Ort) Die Integer-Variable Ortlaenge weist nun den Wert 10 auf.
pos (Suchkette, Zeichvar)	Stellt fest, ab welcher Stelle Suchkette in Zeichvar vorkommt. Beispiel: Zeichvar := 'Düsseldorf'; N := pos ('ld', Zeichvar); Die Integervariable N weist nun den Wert 6 auf. Dagegen ist N gleich null bei N := pos ('x', Zeichvar) weil x nicht in Zeichvar enthalten ist.

Programmbeispiel

In folgendem Programm sind die Stringfunktionen und -prozeduren beispielhaft dargestellt.

```
program Stringprozeduren;
var Ort, Vorort: string [20];
    Plz: string [4];
    GesamtOrt: string [44];
    Laenge, Stelle, Leitzahl, Code: integer;

begin
  Ort := 'Köln-Mülheim'; Plz := '4000';
  Laenge := length (Ort); writeln ('Länge: ',Laenge:3);
  Stelle := pos('-',Ort); writeln ('Position: ',Stelle:3);
  Vorort := copy(Ort,Stelle+1,Laenge-Stelle); writeln (Vorort);
  delete (Ort,Stelle,Laenge-Stelle+1); writeln (Ort);
  val(Plz,Leitzahl,Code);
  if Code = 0
    then writeln('PLZ:', Leitzahl:5)
    else writeln ('Fehler im String');
  Leitzahl := Leitzahl + 1000;
  str(Leitzahl:4,Plz);
  Gesamtort := concat(Plz,Ort,'-',Vorort); writeln (Gesamtort);
  insert(' ',Gesamtort,5); writeln (Gesamtort);
end.
```

Untersuchen Sie das Programm. Geben Sie es in den Rechner ein, und testen Sie seine Wirkungsweise.

7.7.6 Sonstige Standardprozeduren und -funktionen

Prozeduren

Prozeduraufruf	Wirkung
delay (N)	Hält das Programm für so viele Millisekunden an, wie N angibt. N ist ein Integer-Wert.
error (Nr, Adr)	Verfügbar ab Turbo-Pascal 3.0. Tritt zur Laufzeit des Programms ein Fehler auf, so wird in der Integer-Variablen Nr die Fehlernummer und in der Integer-Variablen Adr die interne Speicheradresse vermerkt.
exit	Veranlaßt den Aussprung aus dem aktuellen Programmblock. Wird exit in einem Unterprogramm aufgerufen, wird das Unterprogramm verlassen. Beim Aufruf von exit im Hauptprogramm wird der Programmlauf abgebrochen.

Prozeduraufruf	Wirkung
GetDate	Stellt das aktuelle Systemdatum fest (s. Beispielprogramm).
halt	Bewirkt den Abbruch des Programms.
randomize	Schaltet den Zufallszahlengenerator ein.

Programmbeispiel

```
program WochentagErmittlung;
uses crt, dos;                                    (* ab Version 4 *)
var jahr, monat, tag, wochentag: word;

begin
  ClrScr;
  getdate(jahr, monat, tag, wochentag);
  case wochentag of
    0 : write ('Sonntag');
    1 : write ('Montag');
    2 : write ('Dienstag');
    3 : write ('Mittwoch');
    4 : write ('Donnerstag');
    5 : write ('Freitag');
    6 : write ('Samstag');
  end;
  readln;
end.
```

Funktionen

Funktionsaufruf	Datentyp des Eingabewertes	Datentyp des Ausgabewertes	Bedeutung
IOresult		integer	Stellt fest, ob bei einer Eingabe-/Ausgabeoperation ein Fehler aufgetreten ist.
keypressed		boolean	Gibt den Wahrheitswert true aus, wenn eine Taste der Konsole gedrückt wurde.
readkey		char	Liest ein Zeichen aus dem Tastaturpuffer; das Zeichen wird nicht auto-

Funktionsaufruf	Datentyp des Eingabewertes	Datentyp des Ausgabewertes	Bedeutung
			matisch auf dem Bildschirm angezeigt. (S. Beispielprogramm)
random		real	Erzeugt eine Zufallszahl, die größer gleich null und kleiner eins ist.
random (i)	integer	integer	Bildet eine ganzzahlige Zufallszahl, die größer gleich null und kleiner als der Wert i ist.
upcase (z)	char	char	Wandelt einen Kleinbuchstaben in den entsprechenden Großbuchstaben um. Beinhaltet z bereits einen Großbuchstaben, hat die Funktion keine Wirkung.

Programmbeispiel

Jeder Tastenanschlag erzeugt im Tastaturpuffer einen bestimmten Wert bzw. Code. Um ihn aufzunehmen, stehen zwei Byte zur Verfügung. Folgende Tabelle zeigt einen Ausschnitt aus der Zuordnung von Tastatur und Code.

Taste	Tastaturcode	
A	'A'	
Esc	#27	
F1	#0	59
F2	#0	60
Alt O	#0	16
Alt X	#0	45
Alt C	#0	46

Der Wert im 2. Byte wird auch Scan-Code genannt. Näheres finden Sie dazu im rororo Band 8181 «Computerwissen für Einsteiger».

Durch ReadKey wird des Inhalt des Tastaturpuffers byteweise in die angegebene CHAR-Variable übertragen, die dann abgefragt werden kann.

```
program Tastaturabfrage;
uses crt;                                          (* ab Version 4 *)
var taste: char;

begin
  ClrScr;
  write ('Betätigen Sie eine Taste  ');
  taste := readkey;
  case taste of
     #0 : begin                              (* Test auf Funktionstaste *)
            taste := readkey;                 (* Lesen des 2. Byte *)
            case taste of
              #59 : write ('F1-Taste');
              #45 : write ('ALT-X');
              #80 : write ('Pfeil-oben');
            end;
          end;
    #27 : write ('ESC-Taste');
    else  write ('Keine Sondertaste');
  end;
  readln;
end.
```

8 ÜBERSICHT ÜBER VERSCHIEDENE VERSIONEN VON TURBO-PASCAL

8.1 Turbo-Pascal Version 7

8.1.1 Das Bediensystem handhaben

Bedienfunktionen aufrufen
Es stehen Ihnen mehrere Wege zur Verfügung, um Funktionen des Bediensystems aufzurufen.

1. Möglichkeit: Sie rufen die gewünschte Funktion aus einem Menüfenster auf, das Sie über das Hauptmenü aktivieren. Mögliche Vorgehensweisen:

- Sie klicken den Menüpunkt mit der Maus an.
- Sie betätigen die Taste [F10], wählen den betreffenden Menüpunkt mittels [→] oder [←] aus und bestätigen mit [↵] Ihre Wahl.
- Sie aktivieren den Menüpunkt mittels [Alt] und hervorgehobener Buchstabe. Den hervorgehobenen Buchstaben ersehen Sie aus dem Namen des Menüpunktes.

2. Möglichkeit: Sie rufen die gewünschte Funktion über Funktionstasten auf.

3. Möglichkeit: Da in der Version 7 auch die Tastenkommandos der Version 3 integriert sind, können Sie auch diese benutzen.
Innerhalb des Menüfensters aktivieren Sie die Funktion entweder durch

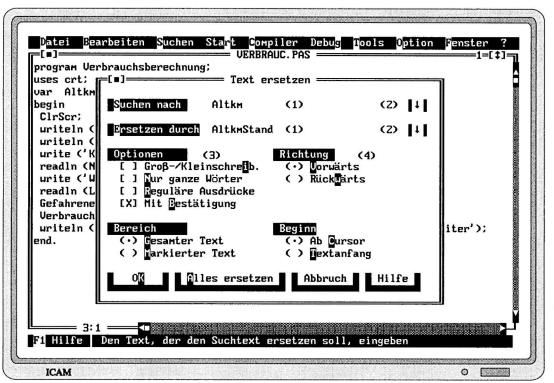
- Anklicken der entsprechenden Menüzeile mit der Maus oder durch
- Auswahl mittels [↑] bzw. [↓] und Bestätigung mit [↵] oder durch

- Eingabe der Tastenkombination Alt und hervorgehobener Buchstabe.

Sie heben die Anwahl eines Menüpunktes durch Eingabe von Esc wieder auf.

Dialogfenster bedienen

Bei einigen Funktionen benötigt das System weitere Angaben. Diese fordert es in sogenannten Dialogfenstern an. Dialogfenster können über Eingabefelder, Schalt- und Markierungsfelder verfügen. Jedes Fenster ist mit Aktionsschaltern ausgestattet.



Sie aktivieren ein Feld mittels der Tastenkombination Alt und hervorgehobener Buchstabe oder durch Anklicken des Feldes mit der Maus. Mit Hilfe der ⇆-Taste sprechen Sie ebenfalls die Felder der Reihe nach an. Die Funktion wird ausgeführt, nachdem Sie den *OK*-Schalter bedient oder die ↵-Taste angeschlagen haben. Die Taste Esc oder der Aktionsschalter *Abbruch* heben ein Dialogfenster wieder auf.

Über Eingabefelder (1) werden dem System Zeichen oder Zeichenfolgen mitgeteilt. Werden vom System Bezeichnungen vorgeschlagen, so können Sie den Vorschlag übernehmen, indem Sie das näch-

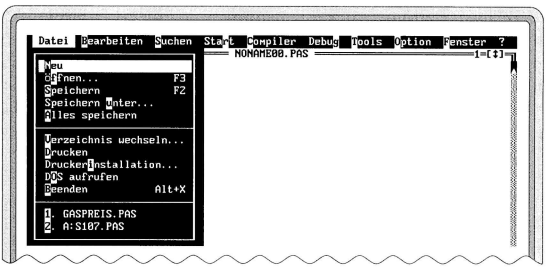
ste Feld ansprechen oder die Funktion ausführen lassen. Falls Sie den Vorschlag verändern möchten, stellen Sie den Cursor mit Hilfe der Tasten [←] bzw. [→] auf die zu ändernde Stelle und nehmen die Änderung vor. Soll der Eintrag auf der ersten Stelle geändert werden, stellen Sie den Cursor mittels [→] und [←] auf die erste Schreibposition und geben anschließend die Änderung ein. Schreiben Sie unmittelbar auf die erste Position, so verschwindet der Systemvorschlag.

Zu jedem Eingabefeld wird eine Eingabeaufzeichnungsliste (2) geführt, in dem die Einträge protokolliert werden, die Sie vorgenommen haben. Sie rufen die Aufzeichnungsliste durch die Taste [↓] oder über das Listensymbol ▮↓▮ auf. Aus dieser Liste können Sie einen Eintrag in das aktuelle Eingabefeld übernehmen, indem Sie den Cursorbalken darauf positionieren und durch [↵] bestätigen.

In einem Markierungsfeld (3) können Einstellungen bzw. Optionen gesetzt oder auch nicht gesetzt werden. Die einzelnen Optionszeilen werden mittels der Pfeiltasten angesteuert. Möchten Sie eine Option markieren oder die Markierung aufheben, so betätigen Sie die Leertaste oder klicken die entsprechende Zeile an.

Schaltfelder (4) sind durch (•) gekennzeichnet. Es kann immer nur eine Einstellung angeschaltet sein. Das Ein- bzw. Ausschalten geschieht mit Hilfe der Pfeiltasten oder durch Anklicken.

8.1.2 Menü *Datei*



Im Menüpunkt *Datei* sind die Funktionen zusammengestellt, die für das Laden, Speichern und Drucken von Quellprogrammen benötigt werden. Damit das System die Dateien im richtigen Verzeichnis sucht bzw. ablegt, geben Sie über den Menüpunkt *Verzeichnis wechseln* das gewünschte Laufwerk und Verzeichnis ein. Über den Menüpunkt *Neu* teilen Sie dem System mit, daß Sie ein neues Programm eingeben möchten. Das System stellt Ihnen dann ein leeres Editfenster mit dem Namen NONAMEXX.PAS zur Verfügung. XX steht für eine Zahl von 00 bis 99. Wenn Sie mittels F2 oder mit Hilfe der Funktion *Speichern* das Programm speichern, fordert das System Sie zur Eingabe eines Dateinamens auf.

Bei *Alles speichern* werden die Inhalte sämtlicher Benutzerfenster, in denen Veränderungen vorgenommen wurden, gespeichert. Die Funktion *Öffnen*, die auch durch F3 aufgerufen wird, lädt eine Datei vom externen Speicher in den Arbeitsspeicher und zeigt sie im aktiven Editfenster an. Möchten Sie ein Quellprogramm noch ein weiteres Mal unter einem anderen Namen speichern, so bedienen Sie sich der Funktion *Speichern unter*.

Der Ausdruck eines Quellprogramms geschieht mittels der Funktion *Drucken*. Sollen die PASCAL-Wörter im Fettdruck erscheinen, müssen Sie vorher im Menü *Druckerinstallation* zum einen den richtigen Druckertyp, zum anderen das Feld *Syntaxhervorhebung* markiert haben.

Möchten Sie nur einen Teil des Programms im aktiven Fenster

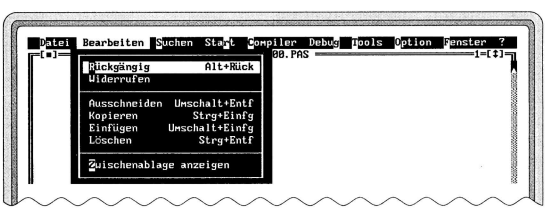
drucken, so markieren Sie ihn und veranlassen den Ausdruck mit Hilfe der Tastenkombination Strg K P.

Sie können Betriebssystemkommandos ausführen lassen, ohne vorher Turbo-Pascal zu beenden. Über die Funktion *DOS aufrufen* wechseln Sie vorübergehend auf die Betriebssystemebene. Es muß jedoch genügend Speicherplatz im konventionellen Arbeitsspeicher verfügbar sein, damit der Kommandoprozessor COMMAND.COM geladen werden kann. Durch Eingabe von exit ↵ kehren Sie in das Pascal-System wieder zurück.

Eine Arbeitssitzung beenden Sie durch Alt X. Dabei werden alle geöffneten Fenster geschlossen. Bei geänderten Dateien fragt das System nach, ob die Änderungen gespeichert werden sollen.

Im unteren Teil des Dateifensters werden die Namen der Dateien, mit denen man während der Arbeitssitzung gearbeitet hat, in einer Liste geführt. Möchten Sie darauf zurückgreifen, so stellen Sie den Cursorbalken auf den entsprechenden Dateinamen und bestätigen Ihre Wahl mittels ↵. Das System aktiviert ein weiteres Fenster, in das es den Inhalt der Datei lädt.

8.1.3 Menü *Bearbeiten*



In dem Menü *Bearbeiten* sind die Funktionen zum Kopieren und Löschen zusammengefaßt. Über die Funktion *Rückgängig*, die Sie auch durch die Tastenkombination Alt ← aktivieren, können Sie frühere Eingaben und Löschungen schrittweise rückgängig machen.

Die Funktion *Ausschneiden*, die auch durch die Tastenkombination ⇧ Entf aktiviert wird, löscht den markierten Text im aktiven Editfenster und kopiert ihn in die *Zwischenablage*. Von dort können Sie

den Text mittels der Funktion *Einfügen* an eine andere Stelle kopieren. Wollen Sie den Textblock duplizieren, so übernehmen Sie ihn mittels *Kopieren* oder durch Strg Einfg in die Zwischenablage. Durch *Löschen* bzw. Strg Entf wird der markierte Block unwiederbringlich gelöscht.
Die Funktion *Zwischenablage anzeigen* schaltet vom Editfenster in das Fenster *Zwischenablage*. Die Einträge dort können Sie in der gleichen Weise bearbeiten wie in einem normalen Editfenster. Ein Kopieren in die Zwischenablage ist jetzt jedoch nicht möglich. Das erkennen Sie auch am Menüfenster *Bearbeiten*. Rufen Sie es aus dem Fenster *Zwischenablage* auf, so sind die Menüpunkte mit verminderter Helligkeit dargestellt, die sich auf das Fenster *Zwischenablage* beziehen. Durch die Darstellung in abgeschwächter Helligkeit macht das System deutlich, daß eine Funktion im gegebenen Zusammenhang nicht verfügbar ist.

8.1.4 Menü *Suchen*

Über das Untermenü *Suchen* rufen Sie die Funktionen auf, die Sie bei Suchoperationen benötigen. Mit den Funktionen *Suchen nach...* und *Ersetzen...* wird der Text im aktiven Editfenster nach bestimmten Zeichen bzw. Zeichenfolgen durchsucht, die Sie über ein Dialogfenster eingeben. Mit der Funktion *Ersetzen...* wird der gesuchte Begriff durch einen von Ihnen angegebenen Begriff ersetzt. Unter welchen Bedingungen der Such- und Ersetzvorgang verlaufen soll, bestimmen Sie in den Optionen des Dialogfensters.
Die Funktion *Gehe zu Zeile...* stellt den Cursor in die Zeile, deren Nummer Sie angeben. Die Funktion *Prozedur suchen* erleichtert in großen Programmen das Auffinden von bestimmten Prozeduren.

Die Funktion *Laufzeitfehler suchen* werden Sie, solange Sie in der integrierten Entwicklungsumgebung arbeiten, nicht benötigen. Hier wird nämlich die Anweisung, bei der ein Laufzeitfehler aufgetreten ist, unmittelbar im Quellprogramm angezeigt. Dagegen kann die Funktion *Letzten Fehler anzeigen* recht hilfreich sein, wenn Sie nach einer Compilierung die fehlerhafte Anweisung im Quellprogramm erneut lokalisieren möchten.

8.1.5 Menü *Start*

Über das Menü *Start* können Sie die Programmausführung im Normal- oder im Testmodus starten. Die Startkommandos enthalten auch eine automatische Compilierung, falls das Quellprogramm noch nicht in einer übersetzten Form vorliegt.

Die Funktion *Ausführen* startet ein Programm im Normalbetrieb. Wollen Sie den Programmlauf beeinflussen, starten Sie das Programm mit einer der Funktionen *Einzelne Anweisung*, *Gehe zur Cursorposition* oder *Gesamte Routine*. Diese Funktionen führen das Programm im Testmodus aus.

Mittels der Funktion *Einzelne Anweisung*, die auch durch F7 aktiviert wird, wird das Programm nach der Ausführung einer Anweisung angehalten. Ein erneutes Betätigen von F7 bewirkt die Ausführung der nächsten Anweisung. Besteht die nächste Anweisung aus dem Aufruf einer Prozedur, die Sie nicht im Einzelschrittverfahren durchlaufen möchten, so betätigen Sie die Taste F8. Die Prozedur wird dann ohne Unterbrechung von außen ausgeführt.

Die Funktion *Gehe zur Cursorposition* ermöglicht es Ihnen, das Programm an einer bestimmten Stelle anzuhalten. Sie stellen dazu den Cursor auf die gewünschte Stelle im Quellprogramm und starten das

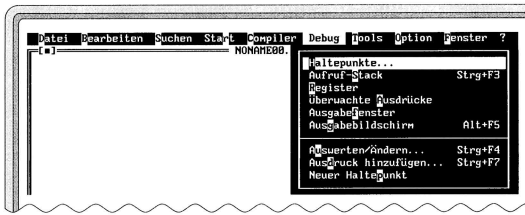
Programm über die Funktion *Gehe zur Cursorposition.* F4 aktiviert ebenfalls diese Funktion.
Mit der Funktion *Programm zurücksetzen* beenden Sie den Programmlauf im Testmodus. Strg F2 bewirkt dies ebenfalls.

8.1.6 Menü *Compiler*

Im Menü *Compiler* sind die Funktionsaufrufe zusammengestellt, mit denen der Übersetzungslauf gestartet wird. Die Funktion *Compilieren* übersetzt das Programm, das sich im aktuellen Editfenster befindet. Diese Funktion rufen Sie ebenfalls mittels Alt F9 auf. Besteht ein Programmsystem aus einem Haupt- und mehreren Teilprogrammen, spricht man von einem Projekt. Die Übersetzung bezieht sich dann auf alle mit dem Projekt verbundenen Teilprogramme. Diese Form der Compiler-Aufrufe wird hier nicht näher behandelt, weil sie im Rahmen einer Einführung ohne Bedeutung ist. Wichtig ist jedoch der Menüpunkt *Ausgabeziel.* Er hat die Funktion eines Schalters. Steht der Schalter auf *Festplatte*, wird bei der Compilierung eine .EXE-Datei in dem angegebenen Verzeichnis erzeugt. Bei der Einstellung *Speicher* wird das Maschinenprogramm lediglich im Arbeitsspeicher aufgebaut. Wollen Sie die Einstellung ändern, wählen Sie den Menüpunkt an und betätigen ↵.

8.1.7 Menü *Debug*

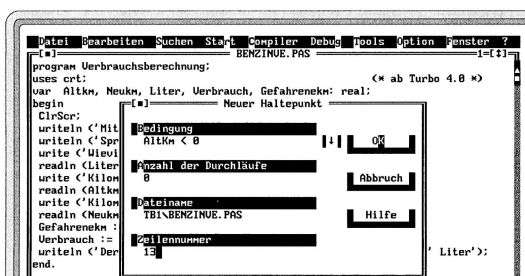
Das Menü *Debug* beinhaltet die Funktionen, die für die Fehlersuche wichtig sind.



Haltepunkte

Beim Testen eines Programms ist es zuweilen sehr zweckmäßig, wenn man das Programm bei Vorliegen einer bestimmten Bedingung anhalten kann. Die Bedingung kann beispielsweise darin bestehen, daß eine Variable einen bestimmten Wert annimmt. Das Überprüfen von Schleifenkonstruktionen wird erleichtert, wenn das Programm erst nach einer vorher festgelegten Anzahl von Durchläufen anhält.

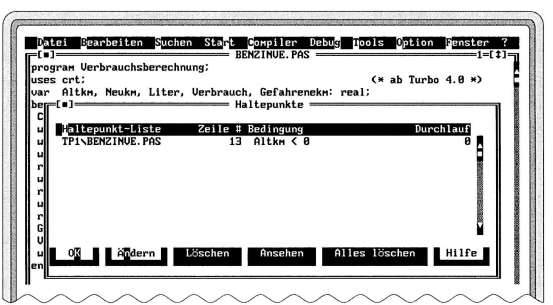
Zum Eintrag eines neuen Haltepunktes betätigen Sie den Aktionsschalter *Neuer Haltepunkt*. Das System blendet daraufhin das Fenster *Neuer Haltepunkt* ein.



In dieses Fenster tragen Sie ein, unter welcher Bedingung und in welcher Zeile das Programm anhalten soll. Wie Sie aus der obigen

Abbildung entnehmen können, würde das Programm in der Zeile 13 stoppen, wenn die Variable AltKm einen Wert kleiner null aufweist. Zum Test von Schleifenkonstruktionen wird in der Zeile *Anzahl der Durchläufe* vermerkt, nach wieviel Durchläufen die Ausführung des Programms unterbrochen werden soll. Im Eingabefeld *Dateiname* wird eingetragen, für welches Programm der Haltepunkt eingerichtet wird. Das System schlägt das aktuelle Programm vor. Im Feld *Zeilennummer* schließlich geben Sie an, in welcher Zeile das Programm anhalten soll, nachdem die Bedingung eingetreten oder die gesetzte Anzahl von Durchläufen erreicht ist. Wurde keine Bedingung oder keine Durchlaufanzahl angegeben, hält das Programm in der hier angegebenen Zeile jeweils an.

Einen Haltepunkt können Sie auch mittels der Tastenkombination Strg F8 einrichten. Dazu stellen Sie den Cursor auf die gewünschte Zeile und betätigen die Tastenkombination. Das System legt daraufhin einen farbigen Balken über die betreffende Zeile. Bei der Programmausführung informiert das System den Nutzer, wenn die Haltebedingung eingetreten ist. Möchten Sie sich die Bedingungen, die für die einzelnen Haltepunkte eingetragen sind, anschauen oder auch verändern, so rufen Sie die Funktion *Haltepunkte* auf. Diese Funktion zeigt die Liste der vereinbarten Haltepunkte an.



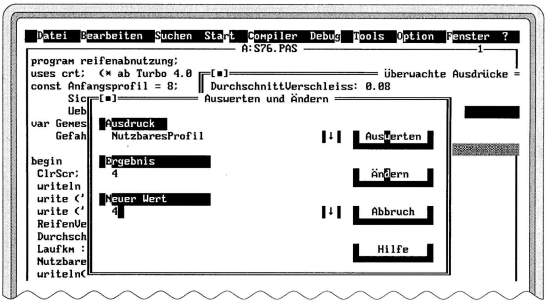
Aus der Haltepunkt-Liste erkennen Sie, in welchem Programm bzw. Modul und in welcher Zeile ein Haltepunkt vereinbart wurde. In

dem obigen Beispiel soll das Programm in Zeile 13 anhalten, wenn die Variable Altkm einen Wert kleiner 0 aufweist. Die Anzahl der Durchläufe spielt hier keine Rolle.
Die Aktionsschalter des Haltepunkt-Fensters bewirken folgende Funktionen:

Aktionsschalter	Funktion
OK	Fenster wird verlassen. Die Einstellungen werden bei der weiteren Programmausführung wirksam.
Ändern	In einem weiteren Fenster können die Angaben zu dem bedingten Haltepunkt verändert werden.
Löschen	Der aktuelle Haltepunkt, gekennzeichnet durch einen Cursorbalken, wird aus der Liste der Haltepunkte gestrichen.
Ansehen	Das System schaltet in das Quellprogrammfenster zurück. Der Cursor steht in der Zeile, die dem aktuellen Haltepunkt im Haltepunkt-Fenster entspricht.
Alles löschen	Sämtliche Haltepunkte werden aufgehoben.
Hilfe	Aufruf des Hilfesystems zu diesem Fenster.

Die Haltepunkte bleiben so lange bestehen, bis sie gelöscht werden. Das Löschen eines Haltepunktes erfolgt aus dem Fenster *Haltepunkte*. Steht im Editfenster der Cursor auf der Zeile mit einem Haltepunkt, so hebt auch die Tastenkombination Strg F8 diesen Haltepunkt wieder auf.

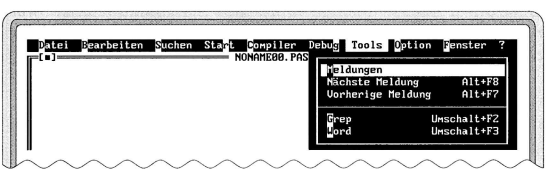
Kontrollfenster
Die Funktion *Aufruf-Stack* wird beim Testen von Programmen gebraucht, die prozedural aufgebaut sind. In einem Fenster wird angezeigt, über welche Prozedur- und Funktionsaufrufe der aktuelle Bearbeitungspunkt des Programms erreicht wurde.
Die Funktion *Register* zeigt in einem weiteren Fenster den Inhalt der CPU-Register an. Diese Angaben können für den Programmierer wichtig sein, der mit dem integrierten Assembler arbeitet.
Die Funktionen *Überwachte Ausdrücke, Ausgabefenster* und *Ausgabebildschirm* aktivieren die entsprechenden Fenster.



Die Funktion *Auswerten/Ändern* bzw. Strg F4 ruft das Fenster auf, über das Inhalte von Variablen angezeigt und verändert werden können. Der Name der betreffenden Variablen ist in dem Feld *Ausdruck* einzutragen. Die Änderung des Inhaltes geschieht im Feld *Neuer Wert.*

Mittels der Funktion *Ausdruck hinzufügen* bzw. Strg F7 stellen Sie eine Variable in das Beobachtungsfenster *Überwachte Ausdrücke.*

8.1.8 Menü *Tools*

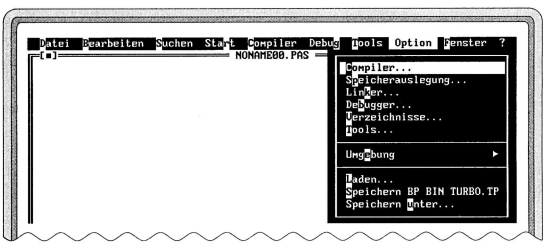


Über das Menü *Tools* können Sie andere Programme aufrufen, ohne dabei die Pascal-Arbeitssitzung beenden zu müssen. Nach Aufruf des anderen Programms, hier zum Beispiel WORD, geht die Systemsteuerung an WORD über. Wenn Sie WORD beenden, so wird die

Pascal-Entwicklungsumgebung wieder aktiviert. Hat das andere Programm Meldungen gesendet, so können Sie sich diese über die Funktion *Meldungen* anzeigen lassen.

Die Programme, die aus der Entwicklungsumgebung aufgerufen werden können, vereinbaren Sie im Menü *Option*.

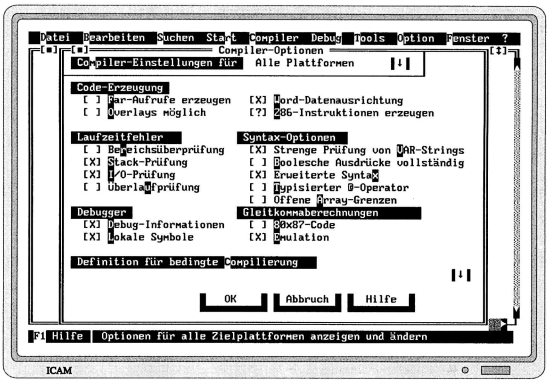
8.1.9 Menü *Option*



Im Menü *Option* wird eine Vielzahl von Parametern und Schaltern eingestellt, die den Compiler, den Linker und das Rechnersystem betreffen. Hier werden nur die Punkte besprochen, die im Rahmen einer strukturierten Einführung von Bedeutung sind.

Compiler

Der Aufruf des Menüpunktes führt zu der Bildschirmmaske *Compiler-Optionen*, in der Sie die Bedingungen für die Erzeugung des Maschinencodes markieren können.



Code-Erzeugung
Soll das von Ihnen geschriebene Programm nur auf Rechnern mit den Prozessoren 80286 und höher laufen, so ist die Einstellung

[X] 286-Instruktionen erzeugen

sinnvoll. Das mit dieser Einstellung compilierte Programm erzielt eine höhere Verarbeitungsgeschwindigkeit. Es läuft jedoch nicht auf Rechnern mit einem 8086-, 80186- oder 8088-Prozessor.

Laufzeitfehler
Ist die Option *Bereichsüberprüfung* markiert, wird vom Compiler ein zusätzlicher Code erzeugt. Das System kann dann während der Programmausführung prüfen, ob bei der Adressierung von Strings, Arrays und Unterbereichen die vereinbarten Grenzen eingehalten werden. Es bricht die Programmausführung ab, wenn beispielsweise versucht wird, auf eine Zeichenkette außerhalb der vereinbarten Grenzen zuzugreifen.
Soll das System automatisch prüfen, ob Eingabe-/Ausgabeoperationen ordnungsgemäß ausgeführt worden sind, so muß die Option *I/O-Prüfung* markiert sein. Wenn Sie nun beispielsweise in eine Integer-

Variable einen Real-Wert eingeben, bricht das Programm mit der Fehlermeldung 106 ab:

Wollen Sie solche Programmabbrüche vermeiden, ist die Option *I/O-Prüfung* auszuschalten. Sie müssen dann jedoch im Programm für eine entsprechende Fehlerbehandlung sorgen. Dafür steht die Funktion **IORESULT** zur Verfügung. IORESULT liefert den Wert 0, wenn eine I/O-Anweisung fehlerfrei durchgeführt wurde. Die Einbindung der Funktion in ein Programm zeigt folgendes Beispiel:

```
program Verbrauchsberechnung;
uses crt;
var Altkm, Neukm, GefahreneKm: longint;
    Liter, Verbrauch: real;

begin
  ClrScr;
  repeat
    gotoxy (5,8);
    write ('Kilometerstand beim vorletzten Tanken?');
    readln (Altkm);
  until IORESULT = 0;
...
```

Diese Programmkonstruktion fängt den Eingabefehler ab. Die Eingabe wird so lange wiederholt, bis IORESULT den Wert 0 ausgibt und damit anzeigt, daß die Operation korrekt ausgeführt wurde. Die Compilerdirektive {$I-} hat dieselbe Wirkung wie die Option *I/O-Prüfung*. Beispiel:

```
...
(*$I-*)
  repeat
    gotoxy (5,8);
    write ('Kilometerstand beim vorletzten Tanken?');
    readln (Altkm);
  until IORESULT = 0;
(*$I+*)
...
```

Mit {$I+} wird die automatische Fehlerprüfung wieder eingeschaltet.

Syntax-Optionen

Im Unterpunkt *Syntax-Optionen* stellen Sie ein, wie sich das System bei der Syntaxprüfung verhalten soll.

Die Syntaxregeln schreiben vor, daß bei der Übergabe von Variablen an eine Prozedur oder Funktion die formalen und aktuellen Parameter hinsichtlich ihrer Datentypen genau übereinstimmen müssen. Liegt keine Typgleichheit vor, bricht das Programm ab. Wird die Option *Strenge Prüfung von VAR-Strings* nicht gesetzt, so erfolgt bei der Übergabe von String-Variablen keine strenge Typprüfung. Das System läßt es dann beispielsweise zu, daß eine vierzigstellige Zeichenkettenvariable übergeben werden kann, auch wenn der formale Parameter nur mit einer Länge von 30 Stellen vereinbart wurde. Dieser Option entspricht die Compileranweisung {$V+} für strenge Prüfung bzw. {$V-} für fehlertolerante Prüfung (vgl. Kapitel 7.4).

In dem Optionsfeld *Boolesche Ausdrücke vollständig* bestimmen Sie, ob Bedingungsausdrücke stets komplett auszuwerten sind, auch wenn der Wahrheitswert schon feststeht. Beispiel:

```
...
until (A=B) and (C<>D) and (E<F)
```

Ist die Option nicht gesetzt, so wird die Bearbeitung des Bedingungsausdrucks beendet, wenn eine Teilbedingung auf false steht.

Debugger

Um in der integrierten Entwicklungsumgebung beim Programmtest den Debugger einsetzen zu können, müssen die beiden Optionen

[X] Debug-Informationen
[X] Lokale Symbole

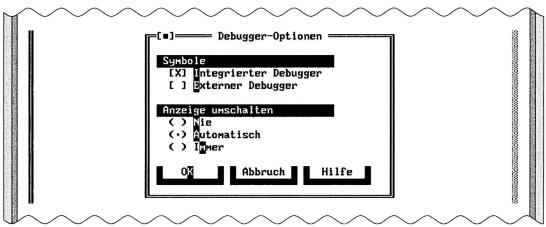
gesetzt sein.

Gleitkommaberechnungen

Im Unterpunkt *Gleitkommaberechnung* geben Sie an, ob bei der Kompilierung der Maschinencode auf einen mathematischen Co-Prozessor ausgerichtet werden soll. Voreingestellt ist *Emulation*. Die

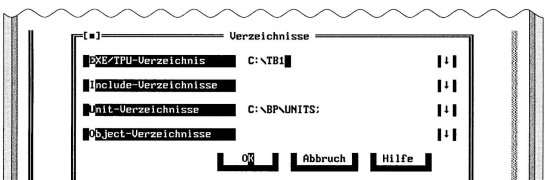
Gleitkomma-Arithmetik wird dann softwaremäßig behandelt. Es wird dann kein Co-Prozessor vorausgesetzt.

Debugger



Die Option *Integrierter Debugger* sollten Sie einschalten, damit Sie beim Programmtest mit dem integrierten Debugger arbeiten können. Die Option *Anzeige umschalten* sollten Sie auf *Automatisch* stellen.

Verzeichnisse



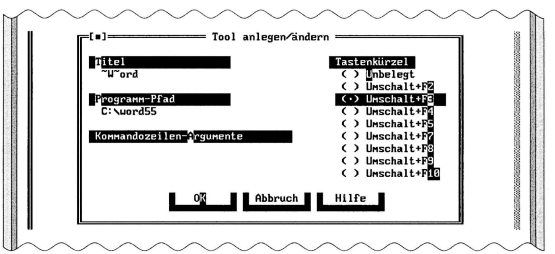
Im Fenster *Verzeichnisse* geben Sie das Verzeichnis an, das die .EXE-Dateien aufnimmt, wenn im Menü *Compiler* als Ausgabeziel *Festplatte* vereinbart wurde.
Verwenden Sie in Ihrem Programm Include-Files, die Sie mittels

{$I Dateiname} einbinden, so tragen Sie in der Zeile *Include-Verzeichnisse* das Laufwerk bzw. das Verzeichnis ein, in dem diese Dateien zu finden sind.

Tools

Im Menüpunkt *Tools* (deutsch: Werkzeuge) können Sie die Programme vereinbaren, die Sie aus der integrierten Entwicklungsumgebung ohne vorherige Beendigung der Pascal-Arbeitssitzung aufrufen möchten. Der Aufruf des Programms erfolgt, wie bereits oben dargelegt, aus dem Hauptmenüpunkt *Tools*.

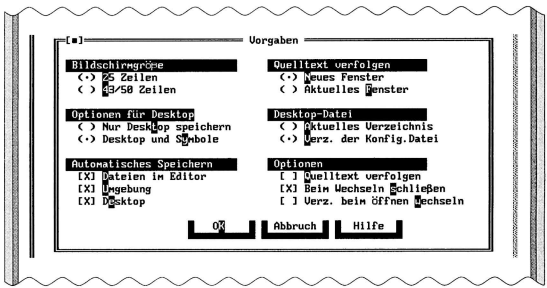
Der Eintrag eines solchen Programmaufrufs sei an einem Beispiel erläutert. In ihm soll das Programm WORD, das im Verzeichnis WORD55 steht, als Werkzeug eingebunden werden. Im Fenster *Tools* betätigen Sie den Schalter *Neu*. Das System schaltet weiter in das Fenster *Tool anlegen/ändern*.



In der Zeile *Titel* tragen Sie den Namen des Programms ein. Wollen Sie das Programm über den Anfangsbuchstaben aufrufen, so setzen Sie diesen zwischen zwei Tilden. Im Feld *Programm-Pfad* geben Sie an, in welchem Verzeichnis sich das Programm befindet. Eventuelle Kommandozeilen-Parameter vermerken Sie in der Zeile *Kommandozeilen-Argumente*. Schließlich stellen Sie in dem Feld *Tastenkürzel* die Tastenkombination ein, über die Sie das Programm aufrufen wollen. Im obigen Beispiel kann das Programm WORD durch Eingabe von W im Menü *Tools* oder durch die Tastenkombination ⇧ F3 aktiviert werden.

Vorgaben/Umgebung

In dem Fenster *Vorgaben* stellen Sie die gewünschte Arbeitsumgebung ein.



Sie erleichtern sich die Arbeit, wenn Sie die drei Optionen im Feld *Automatisches Speichern* aktivieren. Dadurch erreichen Sie, daß beim Verlassen der Entwicklungsumgebung alle geöffneten Dateien, die Sie während einer Arbeitssitzung verändert haben, automatisch gespeichert werden. Außerdem merkt sich das System, welche Dateien Sie bearbeitet haben. Es zeigt sie Ihnen im unteren Teil des Menüfensters von *Datei* in einer Auswahlliste an. Wenn Sie Turbo-Pascal wieder starten, baut das System die Arbeitsumgebung mit all ihren Parametereinstellungen in der Weise wieder auf, wie Sie sie verlassen haben.

Die hierfür erforderlichen Daten werden in der Konfigurationsdatei TURBO.TP und in der Desktop-Datei TURBO.DSK gespeichert. Falls Sie an unterschiedlichen Programmprojekten arbeiten, die in unterschiedlichen Verzeichnissen gespeichert sind, so können Sie für jedes Projekt eine eigene Konfigurations- und Desktop-Datei anlegen. Im Rahmen einer strukturierten Einführung mag es aber ausreichen, wenn die Desktop-Datei in dem Verzeichnis abgelegt wird, in dem auch die Konfigurationsdatei gespeichert wird.

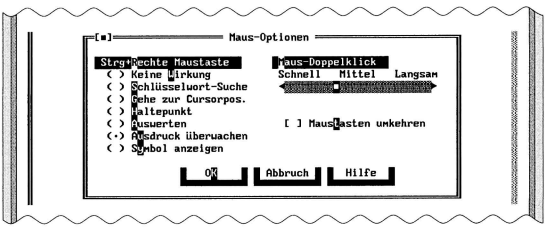
Vorgaben/Editor-Einstellungen

In diesem Fenster legen Sie die Parameter für den Editor fest. Mit dem Parameter *BAK-Dateien anlegen* bestimmen Sie, daß beim Speichern die alte Datei nicht überschrieben wird, sondern als .BAK-Datei erhalten bleibt. Diesen Parameter sollten Sie aus Sicherheitsgründen setzen, es sei denn, daß der Speicherplatz auf dem externen Datenträger knapp ist. Ist der Parameter *Einfügemodus* gesetzt, ist der Editor auf den Einfügemodus voreingestellt. Während des Editierens können Sie aber jederzeit den Schreibmodus durch die Taste Einfg umstellen.

Der Parameter *Autom. Zeileneinzug* erleichtert Ihnen den übersichtlichen Aufbau eines Programms. Bei gesetztem Parameter wird die Einrückung bzw. der Zeileneinzug, den Sie durch die Eingabe von Leerzeichen erzeugt haben, in den Folgezeilen übernommen. Ist der Parameter *Rücktaste löscht Einzug* gesetzt, tabulieren Sie mit der Taste ← rückwärts, wenn sich der Cursor am eingerückten linken Rand befindet.

Weiterhin sollten Sie den Parameter *Persistente Blöcke* (persistent = bleibend) setzen. Dann verschwindet nicht die Markierung eines Blocks, wenn Sie den Cursor bewegen. Der Parameter *Text am Cursor suchen* bewirkt, daß bei Aufruf der Funktionen *Suchen nach* und *Ersetzen* das Wort, das an der Cursorposition steht, automatisch als Suchbegriff in das Dialogfenster übernommen wird.

Vorgaben/Maus

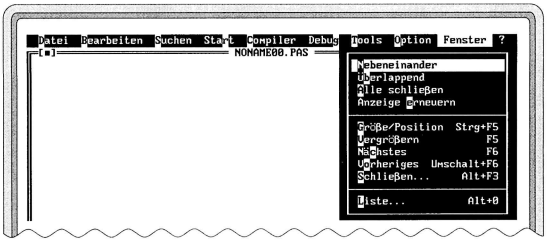


In dem Fenster *Maus-Optionen* legen Sie fest, welche Funktion Sie mit der Bedienkombination von Strg und rechter Maustaste auslösen. Für den Testbetrieb hat es sich als sinnvoll herausgestellt, die Funktion *Ausdruck überwachen* damit zu verbinden.
Am Schiebebalken *Maus-Doppelklick* stellen Sie die Geschwindigkeit ein, mit der Sie einen Doppelklick erzeugen.

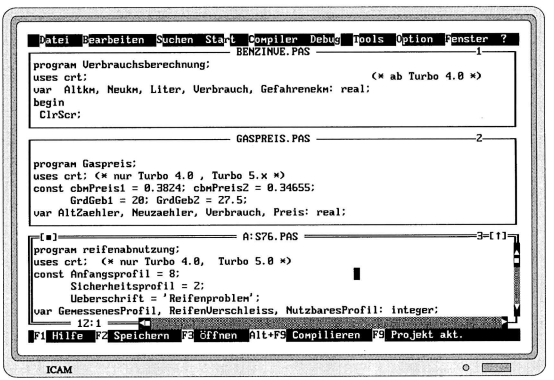
Vorgaben/Voreinstellungen
In diesem Fenster übernehmen Sie die voreingestellten Parameter. Falls Sie Ihr Rechnersystem mit zwei Bildschirmen betreiben, so können Sie dies dem Pascal-System über den Parameter *2 Monitore betreiben* mitteilen. Dann wird auf dem einen Bildschirm die Entwicklerumgebung, auf dem anderen das Anwendungsprogramm angezeigt. Arbeiten Sie an einem Rechner mit einem LCD-Bildschirm, so sollten Sie den Parameter *LCD-Farbpalette* einstellen.

Sonstige Funktionen
Die Funktionen *Laden, Speichern* und *Speichern unter* benötigen Sie nur, wenn Sie mit eigenen Konfigurationsdateien arbeiten.

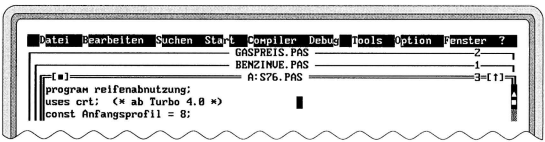
8.1.10 Menü *Fenster*



Im Menü *Fenster* sind die Funktionen zusammengefaßt, die Sie für die Verwaltung von Fenstern benötigen. Ein Großteil der Funktionen wurde bereits in anderem Zusammenhang erläutert. Mittels der Funktionen *Nebeneinander* und *Überlappend* legen Sie fest, wie die Fenster auf dem Bildschirm anzuordnen sind. Die nachstehende Abbildung zeigt die Einstellung *Nebeneinander*.



Ist die Funktion *Überlappend* aktiviert, werden die Fenster versetzt hintereinander angeordnet, wie Sie der nachstehenden Abbildung entnehmen können.



Über die Funktion *Liste* rufen Sie eine Zusammenstellung der zur Zeit geöffneten Fenster auf. Aus dieser Liste können Sie ebenfalls ein bestimmtes Fenster aktivieren.

8.1.11 Menü ?

Der Menüpunkt *?* bietet Ihnen eine weitere Einstiegsmöglichkeit in das Hilfesystem von Turbo-Pascal.

8.1.12 Zusammenfassung

- Zum Aufruf einer Funktion wird der Cursorbalken auf die Menüzeile gestellt und die Taste [↵] betätigt. Dieselbe Wirkung hat das Anklicken der Menüzeile mit der Maus.
- Ein Fenster wird durch Betätigen der Taste [Esc] oder durch Anklicken des Fenstersymbols [■] aufgehoben.
- Zu Eingabefeldern führt das System Aufzeichnungslisten, aus denen Sie frühere Eingaben ersehen sowie auch übernehmen können.
- Wichtige Tastaturkommandos:

Taste	Funktion
[F1]	Hilfesystem aufrufen
[Strg] [F1]	Spracherläuterungen aufrufen
[F3]	Quellprogramm laden
[F2]	Quellprogramm speichern
[Alt] [F9], [F9]	Quellprogramm übersetzen
[Strg] [F9]	Programm starten
[F4]	Programm starten mit Halt bei Cursorposition
[F7], [F8]	Programmstart im Testmodus
[F6], [⇧] [F6]	Fenster wechseln
[Strg] [Einfg]	Block aus Editfenster in die Zwischenablage duplizieren
[Strg] [Entf]	Block aus Editfenster in die Zwischenablage kopieren und Block im Editfenster löschen
[⇧] [Einfg]	Block aus Zwischenablage in das aktuelle Editfenster kopieren
[Strg] [Entf]	Block löschen
[⇧] [F7]	Variable in das Überwachungsfenster eintragen
[Strg] [F2]	Programm zurücksetzen
[Strg] [F4]	Auswerten und Ändern einer Variablen
[Strg] [F8]	Haltepunkt setzen bzw. aufheben
[Alt] [F5]	Ausgabebildschirm aufrufen
[Alt] [F3]	Fenster schließen
[Alt] [X]	Turbo-Pascal beenden

8.1.13 Übungen

1. Aufgabe
Sie möchten Teile eines anderen Programms von der Festplatte in das Programm, das Sie gerade bearbeiten, hineinkopieren. Wie ist zu verfahren?

2. Aufgabe
Sie möchten die Variablenvereinbarung eines Programms für spätere Programmierarbeiten auf die Festplatte schreiben. Welche Schritte sind dazu erforderlich?

3. Aufgabe
Welcher Unterschied besteht zwischen den Haltepunkten, die von F4 und Strg F8 erzeugt werden?

4. Aufgabe
Welche Compileroption muß eingeschaltet sein, damit zum Beispiel die Eingabe einer Real-Zahl in eine Integer-Variable nicht automatisch zum Programmabbruch führt?

5. Aufgabe
Die Compilate sollen auf der Diskette im Laufwerk A: gespeichert werden. Welche Einstellung muß dazu vorgenommen werden?

8.1.14 Lösungen

1. Aufgabe
Öffnen Sie mittels F3 ein neues Fenster, in das Sie das gewünschte Programm von der Festplatte laden. Markieren Sie die zu kopierenden Zeilen und duplizieren Sie sie mittels Strg Einfg in die Zwischenablage. Aktivieren Sie das Fenster, in dem das zu bearbeitende Programm steht. Stellen Sie den Cursor an die aufnehmende Stelle, und betätigen Sie die Tastenkombination ⇧ Einfg.

2. Aufgabe
1. Schritt: Sie markieren die Variablenvereinbarung als Block.
2. Schritt: Durch Strg K W aktivieren Sie die Funktion *Schreiben Block in Datei*.

3. Schritt: In das Eingabefeld des Dialogfensters tragen Sie den Namen der Datei ein, unter der Sie die Vereinbarung speichern möchten.

3. Aufgabe

F4 erzeugt einen temporären Haltepunkt. Der durch Strg F8 eingestellte Haltepunkt bleibt so lange bestehen, bis er aufgehoben wird.

4. Aufgabe

Die Option *I/O-Prüfung* im Menü *Compiler* darf nicht gesetzt sein. Es muß dann aber mittels der Funktion IORESULT für eine ordnungsgemäße Fehlerbehandlung gesorgt werden.

5. Aufgabe

1. Im Menü *Compiler* muß als Ausgabeziel *Festplatte* vermerkt sein.
2. Im Dialogfenster *Verzeichnisse*, das über *Option/Verzeichnisse* geöffnet wird, muß in dem Eingabefeld *EXE/TPU-Verzeichnis* a:\ eingetragen sein.

8.2 Turbo-Pascal Version 6

Das Bediensystem von Turbo-Pascal Version 6 stellt Ihnen mehrere Möglichkeiten zur Verfügung, Funktionen oder Kommandos aufzurufen. In der Regel erfolgt der Aufruf über das Hauptmenü. Einige Funktionen können Sie aber auch direkt über die Funktionstasten aktivieren. Im folgenden werden die wichtigsten Kommandos bzw. Funktionen behandelt, sofern sie nicht bereits in den vorhergehenden Kapiteln angesprochen wurden. Vor ihrer Erläuterung werden Menüaufruf und Eingabeweise bei Dialogfenstern systematisiert dargestellt.

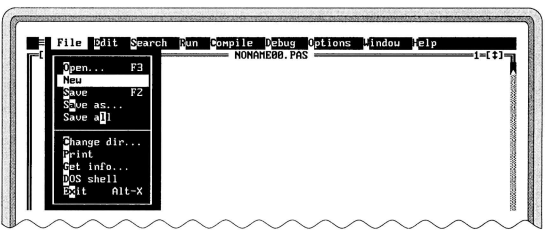
8.2.1 Menü *System*

Das System-Menü umfaßt drei Punkte. Über den Menüpunkt *About* werden Copyright-Informationen angezeigt. Die Funktion *Refresh Display* bewirkt den Neuaufbau des Turbo-Pascal-Bildschirms. Mit-

tels *Clear Desktop* werden alle Fenster und Aufzeichnungslisten geschlossen.

8.2.2 Menü *File*

Der Menüpunkt *File* enthält die Dateiverwaltungsfunktionen.

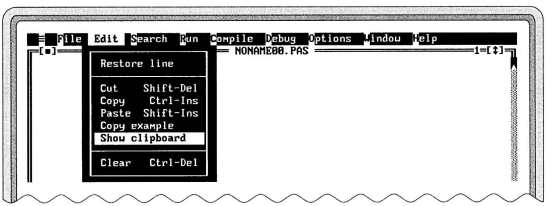


Mittels *Open* bzw. F3 wird eine Datei vom externen Speicher in das aktive Editfenster geladen. Bei Aufruf von *New* wird ein neues Editfenster mit dem Namen NONAMEXX.PAS eröffnet. XX steht für eine Zahl von 0 bis 99. Beim Speichern einer NONAME-Datei werden Sie aufgefordert, die Datei umzubenennen, da diese Datei nur als temporäre, also kurzfristig gültige Datei angesehen wird.
Zum Speichern stehen Ihnen drei verschiedene Speicherkommandos zur Verfügung. Die Funktion *Save* bzw. F2 speichert den Inhalt des aktiven Fensters in einer Datei auf der Festplatte oder Diskette. Mittels *Save as* können Sie das Programm im aktiven Fenster unter einem anderen Dateinamen und/oder in einem anderen Verzeichnis/Laufwerk ablegen. Bei *Save all* werden sämtliche aufgerufenen und veränderten Fensterinhalte gespeichert.
Den Laufwerks-/Verzeichniswechsel vollziehen Sie mittels der Funktion *Change dir. Print* besorgt den Ausdruck der im aktiven Fenster stehenden Datei. Wollen Sie nur einen Teil des Programms im aktiven Fenster drucken lassen, so markieren Sie ihn und verwenden als Druckkommando die Tastenkombination Strg K P.
Über die Funktion *DOS shell* wechseln Sie vorübergehend von Turbo-Pascal auf die Betriebssystemebene. Voraussetzung ist jedoch, daß noch genügend Speicherplatz frei ist, um COMMAND.COM la-

den zu können. Durch Eingabe von exit ↵ kehren Sie in das Pascal-System zurück. Mittels *Exit* (deutsch: Ausgang) bzw. Alt X beenden Sie die Arbeitssitzung und schalten zurück auf die DOS-Ebene.

8.2.3 Menü *Edit*

Das Menü *Edit* beinhaltet Funktionen zum Kopieren und Löschen.



Mit *Restore line* (deutsch: Zeile wiederherstellen) heben Sie das letzte Editorkommando wieder auf.

Beispiel:
Sie haben mit Strg Y eine Zeile gelöscht. Diese Löschung machen Sie mit *Restore line* rückgängig, wenn Sie zwischenzeitlich noch kein anderes Editorkommando gegeben haben.

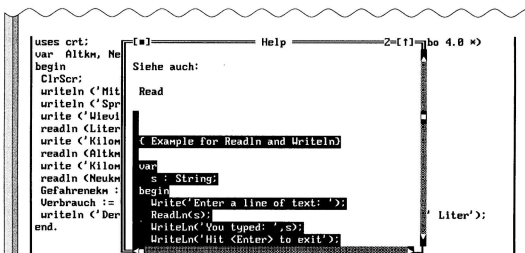
Kopier- und Löschfunktionen
Die Funktionen *Cut, Copy* und *Paste* übertragen markierte Texte zwischen dem aktiven Editfenster und der Zwischenablage bzw. dem Clipboard (deutsch: Klemmbrett). Clipboard ist ein Fenster, das vom System für Kopieroperationen zur Verfügung gestellt wird. *Copy* erzeugt eine Kopie des markierten Textes im Clipboard. Auch *Cut* (deutsch: schneide) kopiert den markierten Text in das Clipboard, löscht ihn aber aus dem aktiven Fenster. Eine unwiderrufliche Löschung bewirkt die Funktion *Clear*. Mit der Funktion *Paste* (deutsch: kleistern) wird der im Clipboard markierte Text an die Cursorposition des aktiven Fensters kopiert. Mittels der Funktion *Show clipboard* (deutsch: zeige Clipboard) aktivieren Sie das Clipboard, in dem Sie in derselben Weise editieren können wie in einem Editfenster.

Kopieren von Beispielprogrammen

Die Funktion *Copy example* (deutsch: kopiere Beispiel) hat ihre Bedeutung im Rahmen des Hilfesystems.

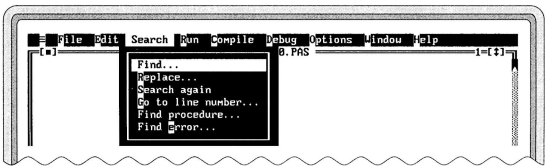
Beispiel:

Sie möchten sich während der Programmeingabe ganz genau über die Syntax des Befehls readln informieren. Stellen Sie den Cursor auf das Wort readln, und betätigen Sie die Tastenkombination Strg F1. Das System reagiert mit der Anzeige eines Hilfefensters. Bestandteil dieses Hilfefensters ist auch ein Beispielprogramm. Wenn Sie dieses Programm nutzen möchten, rufen Sie die Funktion *Copy example* auf. Dadurch wird das Programm in das Clipboard kopiert und steht Ihnen dort zur Nutzung zur Verfügung.



8.2.4 Menü *Search*

Das Menü *Search* (deutsch: suchen) besteht aus Funktionen zum Suchen sowie Ersetzen im aktiven Editfenster.



Mit der Funktion *Find* können Sie eine bestimmte Zeichenfolge oder ein Zeichenmuster im Text des aktiven Editfensters suchen. Bei der Funktion *Replace* wird der gesuchte Text durch einen anderen Text ersetzt. Durch *Search again* wird das letzte Find- bzw. Replace-Kommando wiederholt. Die Funktion *Go to line number* stellt den Cursor in eine vom Benutzer angegebene Zeile. Die Funktion *Find procedure* sucht zu einem eingegebenen Prozedurnamen den Prozedurkopf.

8.2.5 Menü *Run*

Das Menü *Run* enthält die Kommandos, mit denen ein Programm im Normalbetrieb oder im Trace-Betrieb gestartet werden kann. Die Startkommandos enthalten auch einen Compileraufruf. Liegt das Quellprogramm noch nicht in einer übersetzten Form vor, so wird das Programm zunächst kompiliert.

Das Kommando *Run* bzw. Strg F9 startet das Programm für den Normalbetrieb. Damit wird das Programm bis zum Ende in einem Zug ausgeführt, es sei denn, Sie haben einen Haltepunkt in das Programm eingebaut. Mit dem Kommando *Go to cursor* wird das Pro-

gramm bis zu der Zeile ausgeführt, die durch die Cursorstellung angegeben wird.

Trace: Schrittweise Programmausführung
Bei *Trace into* und *Step over* wird das Programm jeweils schrittweise abgearbeitet. Durch [F7] bzw. [F8] wird die Anweisung, auf der der Tracebalken steht, ausgeführt. Zwischen diesen beiden Kommandos gibt es jedoch einen wesentlichen Unterschied. Arbeiten Sie im *Step-over*-Modus, so werden die Unterprogramme nicht schrittweise, sondern ohne Unterbrechung ausgeführt. Ist ein Unterprogramm zu testen, so müssen Sie die Form *Trace into* einsetzen. Hierbei wird jede Anweisung, die innerhalb einer Prozedur liegt, schrittweise ausgeführt.
Beim Debugging ist es wichtig, den Programmlauf ordnungsgemäß abbrechen zu können, wenn man den Fehler gefunden hat. Dazu dient das Kommando *Program reset* bzw. [Strg] [F2].

8.2.6 Menü *Compile*

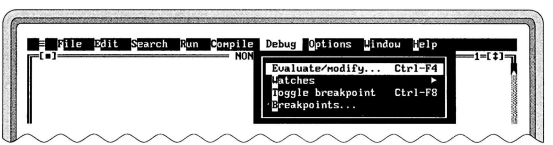
Im Menü *Compile* sind die Kommandos zusammengestellt, die für das Übersetzen von Programmen erforderlich sind.

Das Kommando *Compile* übersetzt das Programm, das sich im aktuellen Editfenster befindet.

Destination
In Abhängigkeit von dem Schalter *Destination* (deutsch: Bestimmungsort) wird das Compilat im Arbeitsspeicher oder als .EXE-Datei auf dem externen Speicher abgelegt. Die Standardeinstellung steht auf *Memory*. Wenn Sie ↵ betätigen oder das Feld anklicken, wird *Destination* auf *Disk* umgestellt.
Steht *Destination* auf *Disk*, wird das Compilat in dem Verzeichnis, in dem das Quellprogramm steht, gespeichert. Haben Sie jedoch im Menüpunkt *Options/Directories* für .EXE-Dateien ein anderes Verzeichnis angegeben, so wird das Compilat dorthin geschrieben. Als Dateiname wird der des Quellprogramms verwendet. Der Zusatz lautet jetzt nicht mehr .PAS, sondern .EXE. So wird beispielsweise das Compilat zu VERBRAUC.PAS in die Datei VERBRAUC.EXE geschrieben.
Der EXE-Zusatz besagt, daß das Programm unmittelbar vom Betriebssystem ausgeführt werden kann. Es wird von der Betriebssystemebene gestartet. Zu seiner Ausführung wird also keine Turbo-Pascal-Umgebung benötigt.

8.2.7 Menü *Debug*

Über das Menü *Debug* werden die Funktionen aufgerufen, die für die Inspektion und Manipulation von Variablen benötigt werden.



Variablen inspizieren und modifizieren
Mit *Evaluate/modify* (deutsch: auswerten/verändern) können Sie sich Inhalte von Variablen anzeigen lassen und sie ändern.
Die Anwahl des Punkts *Watches* blendet ein weiteres Fenster ein, über das die verschiedenen Überwachungs- bzw. Watchfunktionen aufgerufen werden. Das Kommando *Add watch* öffnet ein Fenster, in dessen Eingabezeile Sie den Namen der Variablen eintragen können,

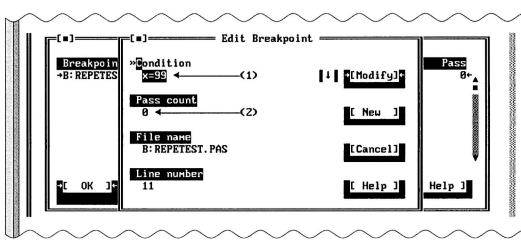
die Sie beobachten wollen. Beim Aufruf des Fensters wird der Begriff, der sich auf der Cursorposition im Editfenster befindet, vorgeschlagen. Mittels *Edit watch* verändern Sie den aktuellen Ausdruck. *Delete watch* löscht den aktuellen Ausdruck im aktiven Watchfenster. Möchten Sie sämtliche Einträge in dem Watchfenster löschen, so rufen Sie die Funktion *Remove all watches* auf.

Haltepunkte setzen

Bei den Programmunterbrechungspunkten werden drei verschiedene Formen unterschieden. Der temporäre Haltepunkt ist durch die Cursorposition gegeben, wenn das Programm mit dem Startkommando *Go to cursor* bzw. F4 gestartet wird. *Toggle breakpoint* bzw. Strg F8 erzeugt einen unbedingten Haltepunkt. Die Anweisung, auf die ein Unterbrechungspunkt gesetzt wurde, wird hervorgehoben dargestellt. Die erneute Anwendung dieser Funktion auf einen Haltepunkt hebt den Haltepunkt auf.

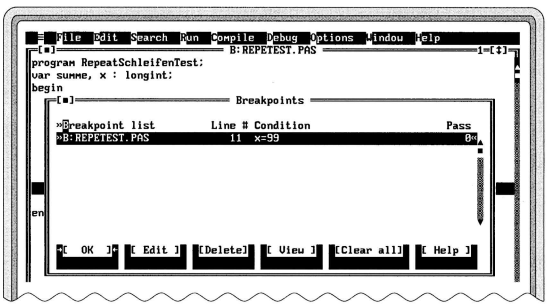
Wenn Sie das Programm mit Strg F9 starten, läuft das Programm bis zu dem Haltepunkt. Sie können dann Variablen inspizieren oder verändern. Das Programm setzen Sie mit Strg F9 fort. Mit Strg F2 brechen Sie den Programmlauf ab.

Schließlich können Sie bedingte Haltepunkte setzen. Dazu tragen Sie in dem Editfenster zu *Breakpoints* ein, in welcher Zeile und unter welcher Bedingung das Programm angehalten werden soll.



Ist beispielsweise ein Haltepunkt in der Zeile 11 mit der Bedingung x = 99 gesetzt, so wird die Programmausführung in Zeile 11 angehalten, wenn x den Wert 99 aufweist. Sie können den bedingten Halte-

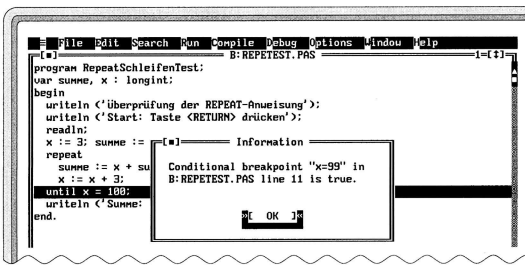
punkt über *Pass count* auch in der Weise formulieren, daß eine Anweisung n-mal bearbeitet werden soll, bis das Programm anhält.
Mit dem Aktionsschalter *New* (4) wird ein neuer Haltepunkt beschrieben. Um die Angaben zu einem bereits bestehenden zu ändern, benutzt man *Modify* (3). Die Eintragungen zu den bedingten Haltepunkten werden im *Breakpoints*-Fenster angezeigt.



Über die Aktionsschalter des *Breakpoints*-Fensters werden folgende Aktionen aufgerufen:

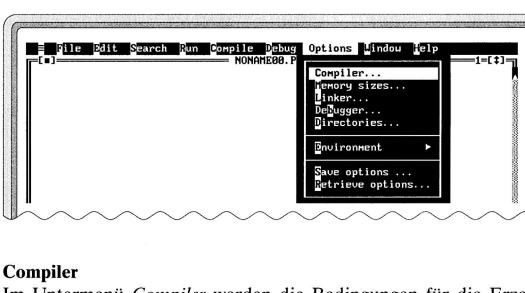
Aktionsschalter	Funktion
OK	Das Fenster wird verlassen. Die Einstellungen werden bei der weiteren Programmausführung wirksam.
Edit	In einem weiten Fenster können die Angaben zu dem bedingten Haltepunkt eingegeben werden.
Delete	Der aktuelle Haltepunkt, gekennzeichnet durch einen Cursorbalken, wird aus der Liste der Haltepunkte gelöscht.
View	Zu dem aktuellen Haltepunkt wird die Stelle im Quelltext angezeigt.
Clear all	Sämtliche Haltepunkte werden aufgehoben.
Help	Aufruf der Hilfefunktion zu diesem Fenster.

Bei der Ausführung des Programms informiert das System den Nutzer, wenn die Haltebedingung erreicht wurde.



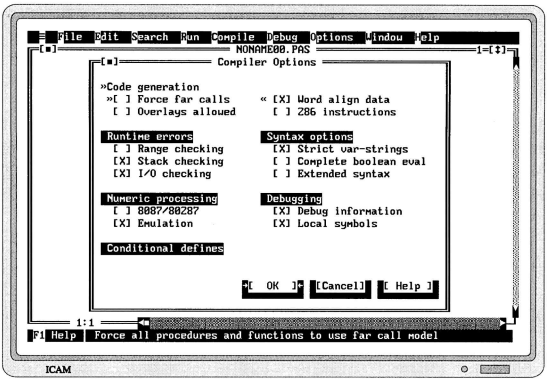
8.2.8 Menü *Options*

Im Menü *Options* wird eine Vielzahl von Schaltern und Parametern gesetzt, die den Compiler, den Linker und das Rechnersystem betreffen. Hier werden nur die Punkte besprochen, die im Rahmen einer strukturierten Einführung von Bedeutung sind.



Compiler

Im Untermenü *Compiler* werden die Bedingungen für die Erzeugung des Maschinencodes markiert.



(1) Berücksichtigung des Prozessortyps bei der Codegenerierung
Verfügen Sie über einen AT-Rechner, so ist möglicherweise die Option *286 instructions* sinnvoll. Beim Übersetzen wird dann ein auf die Prozessortypen 80286, 80386 und 80486 ausgerichteter Maschinencode generiert. Das Maschinenprogramm erzielt zwar eine höhere Verarbeitungsgeschwindigkeit, es läuft jedoch nicht auf einem PC- oder XT-Rechner.
(2) Behandlung von Laufzeitfehlern
Ist das Feld *Range checking* (deutsch: Bereichsprüfung) markiert, so erzeugt der Compiler einen zusätzlichen Code. Während der Programmausführung kann das System dann prüfen, ob bei der Adressierung von Strings, Arrays und Unterbereichen die vereinbarten Grenzen eingehalten werden. Das Programm bricht ab, wenn beispielsweise versucht wird, auf einen String außerhalb der vereinbarten Grenzen zuzugreifen.
Bei gesetzter Option *I/O checking* (deutsch: Eingabe-/Ausgabeprüfung) wird ein Code erzeugt, der nach jeder Eingabe-/Ausgabeoperation prüft, ob die Operation ordnungsgemäß verlaufen ist. Wurde beispielsweise in eine Integer-Variable ein Real-Wert eingegeben, so bricht das Programm ab mit der Fehlermeldung 106: Falsches numerisches Format:

Um solche Programmabbrüche zu vermeiden, wird die Option nicht markiert. Im Programm muß dann jedoch eine Fehlerbehandlung vorgesehen sein. Dafür stellt das System die Funktion IORESULT zur Verfügung. IORESULT liefert den Wert 0, wenn eine I/O-Anweisung fehlerfrei durchgeführt wurde.

Beispiel:

```
program Verbrauchsberechnung;
uses crt;
var Altkm, Neukm, GefahreneKm: longint;
    Liter, Verbrauch: real;

begin
  ClrScr;
  writeln ('Mit diesem Programm wird der durchschnittliche');
  writeln ('Spritverbrauch auf 100 km berechnet');
  repeat
    gotoxy (5,8);
    write ('Kilometerstand beim vorletzten Tanken?');
    readln (Altkm);
  until IORESULT = 0;
...
```

Bei dieser Programmkonstruktion wird ein Eingabefehler abgefangen. Die Eingabe wird so lange wiederholt, bis aus IORESULT zu entnehmen ist, daß die Operation korrekt ausgeführt wurde. Die Compilerdirektive (*$I–*) hat dieselbe Wirkung wie *I/O checking*.

Beispiel:

```
...
(*$I-*)
  repeat
    gotoxy (5,8);
    write ('Kilometerstand beim vorletzten Tanken?');
    readln (Altkm);
  until IORESULT = 0;
(*$I+*)
...
```

Mit (*$I+*) wird die automatische Fehlerprüfung wieder eingeschaltet.

(3) Einstellungsmöglichkeiten zur Syntax

Im Unterpunkt *Syntax options* können Sie die Verhaltensweise des Systems bei der Syntaxprüfung beeinflussen.

Nach den Syntaxregeln müssen bei der Übergabe von Variablen an eine Prozedur oder Funktion die formalen und aktuellen Parameter hinsichtlich ihrer Datentypen exakt übereinstimmen. Liegt keine Typidentität vor, bricht das Programm ab. Wird die Option *strict var-strings* nicht gesetzt, so wird bei der Übergabe von String-Variablen keine strenge Typprüfung vorgenommen. Das System läßt dann beispielsweise die Übergabe einer dreißigstelligen Zeichenkette zu, auch wenn der formale Parameter mit einer Länge von 60 Stellen vereinbart wurde. Diese Option entspricht der Compileranweisung:

(*$V+*) bzw. (*$V-*).

In dem Markierungsfeld *Complete boolean eval* können Sie bestimmen, ob Bedingungsausdrücke komplett ausgewertet werden sollen, auch wenn der Wahrheitswert bereits feststeht.

(4) Co-Prozessor oder Emulation

Im Unterpunkt *Numeric processing* teilen Sie dem System mit, ob bei der Kompilierung der mathematische Co-Prozessor berücksichtigt werden soll. Standardmäßig ist *Emulation* (deutsch: Nachbildung) eingestellt. Die Realzahl-Arithmetik wird dann softwaremäßig behandelt. Diese Programme setzen keinen Co-Prozessor voraus.

(5) Debugging

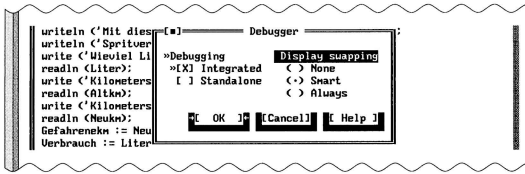
Möchten Sie in der integrierten Entwicklungsumgebung mit dem Debugger arbeiten, so muß beim Kompilieren der für das Debugging erforderliche Code erzeugt werden. Deshalb sind die beiden Felder

[X] Debug information
[X] Local symbols

zu markieren.

8.2.9 Menü *Options/Debugger*

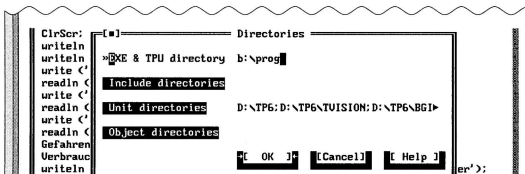
Im Menü *Debugger* sind die für das Debugging erforderlichen Einstellungen vorzunehmen.



Damit Sie Ihr Programm sowohl in der integrierten Entwicklungsumgebung als auch mit dem Debugger bearbeiten können, ist das Feld *Integrated* zu markieren.

8.2.10 Menü *Options/Directories*

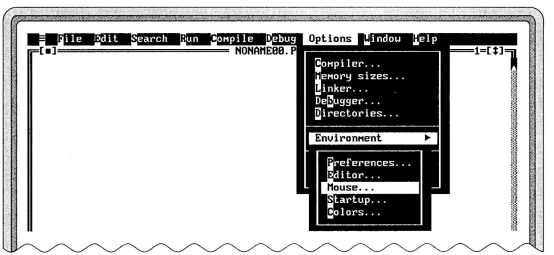
Im Menüpunkt *Directories* geben Sie das Verzeichnis an, das die übersetzten Programme aufnehmen soll, wenn als Destination (Ausgabeziel) *Disk* angegeben wurde. In dem nachstehenden Beispiel wurde als Zielverzeichnis das Verzeichnis PROG auf der Diskette im Laufwerk b: eingetragen.



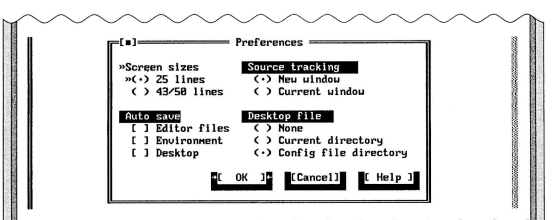
Verwenden Sie in Ihrem Programm Include-Files, die Sie mittels (*$I*) einbinden, so teilen Sie dem System unter *Include directories* mit, in welchem Laufwerk/Verzeichnis diese Dateien zu finden sind.

8.2.10 Menü *Options/Environment*

Im Menü *Environment* (deutsch: Umgebung) werden die Voreinstellungen, die die Arbeitsoberfläche betreffen, vorgenommen.



Preferences
Im Dialogfenster *Preferences* (deutsch: Bevorzugung) können Sie unter anderem die Bildschirmgestaltung beeinflussen.



In *Screen sizes* (deutsch: Bildschirmgröße) können Sie, falls an Ihren Rechner ein EGA- oder VGA-Monitor angeschlossen ist, die Anzahl der Zeilen auf dem Bildschirm auf 43 bzw. 50 erhöhen. Ist das Feld *Editor files* markiert, so speichert das System beim Verlassen von Turbo-Pascal automatisch das Programm, das sich im Editfenster

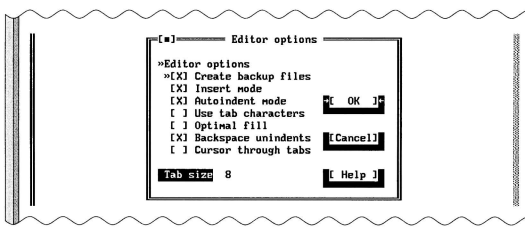
befindet und seit dem letzten Speichern geändert wurde. Der Umfang der automatischen Speicherung bezieht sich auf alle geöffneten Fenster, wenn Sie *Desktop* (deutsch: Arbeitstischoberfläche) markiert haben.

Falls die Einstellungen, die im Menü *Environment* getroffen worden sind, automatisch gespeichert werden sollen, müssen Sie die Option *Environment* setzen.

Im Unterpunkt *Desktop file* legen Sie fest, ob und wo die Informationen zur Arbeitsoberfläche gespeichert werden sollen. Zu den Informationen gehören unter anderem die Einstellungen des Editfensters, der Inhalt der Aufzeichnungsliste und die Position der Breakpoints. Die Speicherung erfolgt in der Datei TURBO.DSK.

Editor

Im Fenster *Editor options* werden die Voreinstellungen für den Editor festgelegt.



Ist das Feld *Create backup files* (deutsch: erzeuge Sicherungsdateien) markiert, wird beim Save-Kommando automatisch eine Sicherungsdatei mit der Endung .BAK erzeugt. Diese Einstellung ist sinnvoll, es sei denn, daß der Speicherplatz auf dem externen Datenträger sehr knapp ist.

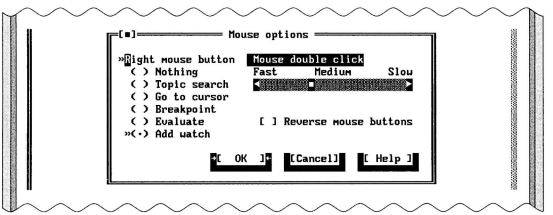
Die Markierung im Feld *Insert mode* besagt, daß der Editor auf den Einfügemodus voreingestellt ist. Beim Editieren schalten Sie durch Einfg in den Überschreibmodus und umgekehrt.

Damit beim Eingeben des Programmtextes automatisch die Einrückung, die Sie mittels der Leertaste erzeugt haben, in den folgenden Zeilen berücksichtigt wird, muß der *Autoindent mode* (deutsch:

automatische Einrückung) eingeschaltet sein. Ist *Backspace unindents* markiert, so tabulieren Sie mit [←] rückwärts, wenn der Cursor am eingerückten linken Rand steht.
In dem obigen Bildschirmabzug ist *Tab size* (Tabulatorgröße) mit 8 angegeben. Beim Betätigen der Tabulatortaste springt der Cursor um 8 Stellen weiter. Die Sprungweite kann in einem Eingabefenster umgestellt werden. Zulässig sind Werte im Bereich von 2 bis 16.

Mouse

In dem Fenster *Mouse options* können Sie entscheiden, ob Sie die linke oder die rechte Maustaste als Auslösetaste benutzen wollen. Die Voreinstellung steht auf linke Taste. Wollen Sie die Voreinstellung ändern, markieren Sie das Feld *Reverse mouse buttons* (deutsch: umgekehrter Mausschalter).



Ferner können Sie festlegen, mit welcher Funktion die rechte bzw. die andere Maustaste verbunden sein soll. Wenden Sie häufig Trace und Debugging an, empfiehlt sich die Verbindung mit *Add Watch*. Dann ist das Eintragen einer Variablen in das Beobachtungsfenster sehr einfach. Sie stellen während der schrittweisen Programmausführung den Mauszeiger auf die gewünschte Variable und betätigen die rechte Maustaste. Damit wird die Variable in das Watchfenster übernommen.
Über den Schiebebalken *Mouse double click* stellen Sie ein, ob Sie sich viel oder wenig Zeit lassen wollen, um einen Doppelklick zu erzeugen.

8.2.12 Menü *Window*

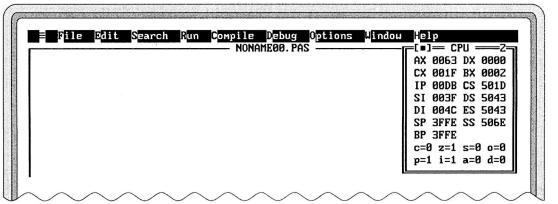
Das Menü *Window* umfaßt all die Funktionen und Kommandos, mit denen Fenster aufgerufen, verändert und geschlossen werden.

Fenster ändern

Über die Funktionen *Size/Move* und *Zoom* wird die Größe und Lage eines Fensters verändert. Der Aufbau der Fenster am Bildschirm wird durch *Tile* (engl. Kachel) oder *Cascade* beeinflußt. Bei der Einstellung *Tile* werden die verschiedenen Fenster übereinander angelegt. Bei *Cascade* werden sie überlappend dargestellt. Mittels *Next* schalten Sie auf das nächste Fenster, mit *Previous* auf das vorhergehende. *Close* schließt das aktive Fenster.

Fenster öffnen

Mit *Watch, Output* sowie *Register* werden neue Fenster geöffnet. Über die Funktion *Register* erhalten Sie einen Einblick in den Inhalt der CPU-Register. Diese Information kann wichtig sein, wenn man mit dem integrierten Assembler arbeitet.



Call stack
Die Funktion *Call stack* zeigt in einem Fenster an, über welche Prozedur- und Funktionsaufrufe der aktuelle Bearbeitungspunkt des Programms erreicht wurde. Beim Testen komplexer Programme ist das eine große Hilfe, um den Programmablauf nachvollziehen zu können.

Window List
In der Fensterliste wird angezeigt, welche Fenster geöffnet sind. Aus dieser Liste heraus können Sie Fenster schließen. Dazu stellen Sie den Cursorbalken auf das betreffende Fenster und betätigen den Aktionsschalter *Delete*.

8.2.13 Menü *Help*

Im Menü *Help* sind die Aufrufe von Hilfefenstern, auf die Sie auch aus anderen Menüs zugreifen können, zusammengefaßt. Der Aufruf erfolgt dort in der Regel mittels F1. Sprachbezogene Hilfen werden über Strg F1 aufgerufen.

Contents (deutsch: Inhalt) zeigt das Inhaltsverzeichnis des Hilfesystems an. Aus diesem Verzeichnis heraus können Sie in jedes Sachgebiet verzweigen. Über *Index* rufen Sie die Liste der Schlüsselwörter auf. Die Hilfen bezüglich der Pascal-Sprache erhalten Sie über *Topic search* (deutsch: themenbezogene Suche).
Auf das jeweils vorletzte Hilfefenster schalten Sie mittels *Previous topic* um. *Help on help* erläutert den Umgang mit dem Hilfesystem.

8.2.14 Zusammenfassung

- Zu jeder Eingabeposition bietet das Turbo-Pascal-System Hilfen an, die über F1 abgerufen werden können.
- Mittels Esc wird ein Menü oder Dialogfenster aufgehoben.
- Wichtige Kommandos:

Funktion	Aufruf
Quellprogramm laden	F3
Quellprogramm speichern	F2
Quellprogramm übersetzen	Alt F9, F9
Programm starten	Strg F9
Programm starten mit Halt bei Cursorposition	F4
Programm zur schrittweisen Ausführung starten	F7, F8
Fenster wechseln	F6, ⇧ F6
Block aus Editfenster ins Clipboard kopieren	Strg Einfg
Block aus Editfenster ins Clipboard kopieren mit Löschen des Blocks im Editfenster	⇧ Entf
Block aus Clipboard ins Editfenster kopieren	⇧ Einfg
Block löschen	Strg Entf
Variable in das Beobachtungsfenster eintragen	⇧ F7
Inspektion und Modifikation von Variablen	Strg F4
Haltepunkt setzen bzw. aufheben	Strg F8
DOS-Bildschirm aufrufen	Alt F5
Fenster schließen	Alt F3
Turbo-Pascal beenden	Alt X

Soll das Compilat auf der Platte oder Diskette gespeichert werden, so muß *Compile/Destination* auf *Disk* stehen. Das Zielverzeichnis kann in *Options/Directories/EXE & TPU-directory* angegeben werden.

8.2.15 Übungen

1. Aufgabe
Sie haben ein Programm eingegeben und möchten es nun speichern. Wie ist zu verfahren?

2. Aufgabe
Wie lautet das kürzeste Kommando zum Laden eines Quellprogramms?

3. Aufgabe
Sie wollen einen Programmbaustein, den Sie in der laufenden Arbeitssitzung schon einmal aufgerufen und gelöscht haben, erneut vom externen Speicher in das Clipboard laden. Erklären Sie die Vorgehensweise!

4. Aufgabe
Sie möchten ein Programm, das aus der Datei VERSUCH1.PAS gelesen wurde, unter dem Namen VERSUCH2.PAS speichern. Mit welchem Kommando erreichen Sie das?

5. Aufgabe
Nach dem Laden eines Programms wollen Sie den Cursor in die Zeile 162 stellen. Mit welchem Kommando geht das am schnellsten?

6. Aufgabe
Worin besteht der Unterschied zwischen *Trace into* und *Step over*?

7. Aufgabe
Das Compilat soll als .EXE-File im Verzeichnis C:/TURB/UEB abgelegt werden. Welche Einstellungen müssen Sie vornehmen?

8. Aufgabe
Sie möchten beim Programmtest das Programm in der Zeile 42 anhalten. Wie können Sie das erreichen?

9. Aufgabe
Sie möchten verhindern, daß beim wiederholten Speichern eines Quellprogramms eine Sicherungsdatei (.BAK) angelegt wird, da wenig Speicherplatz auf der Diskette frei ist. Wie erreichen Sie das?

10. Aufgabe
Ihr Rechner verfügt über den Co-Prozessor 80287. Das soll der Compiler bei der Erzeugung des Maschinencodes berücksichtigen. Welche Einstellung ist vorzunehmen?

11. Aufgabe
Sie wünschen eine automatische Speicherung des Programms, das sich im aktiven Editfenster befindet, wenn Sie die Arbeitssitzung beenden. Welche Option ist einzustellen?

8.2.16 Lösungen

1. Aufgabe
Mittels F2 aktivieren Sie das Dialogfenster *Save file as*. In die Eingabezeile unter *Save file as* geben Sie den Namen der Datei ein, in der das Programm gespeichert werden soll. Den Zusatz .PAS brauchen Sie nicht einzugeben. Er wird vom System automatisch vergeben.

2. Aufgabe
Das Laden eines Quellprogramms erfolgt mit F3.

3. Aufgabe
Sie schalten über *Edit/Show clipboard* oder F6 in das Clipboard um. Mittels Strg K R aktivieren Sie das Dialogfenster *Read Block from File*. Durch ↓ rufen Sie die Eingabeaufzeichnungsliste auf, aus der Sie den gewünschten Dateinamen auswählen.

4. Aufgabe
Sie verwenden das Kommando *Save as* im Menü *File*.

5. Aufgabe
Um den Cursor in eine bestimmte Zeile zu stellen, verwenden Sie die Funktion *Search/Go to line Number*.

6. Aufgabe
Bei *Step over* werden Prozeduren und Funktionen in einem Zug ausgeführt. Bei *Trace into* wird das Einzelschrittverfahren auch auf die Anweisungen in Prozeduren und Funktionen angewendet.

7. Aufgabe
Sie tragen in *Options/Directories/EXE & TPU directory* **c:/turb/ueb** ein. Im Menü *Compile* muß *Destination* auf *Disk* stehen.

8. Aufgabe
Stellen Sie den Cursor in die Zeile 42. Mittels [Strg] [F8] setzen Sie einen Haltepunkt.

9. Aufgabe
Die Option in *Editor/Options/Create backup files* wird nicht markiert.

10. Aufgabe
Das Markierungsfeld [X] 8087/80287 in *Options/Compiler options* muß markiert sein.

11. Aufgabe
Die Option *Environment/Preferences/Editor files* muß gesetzt sein.

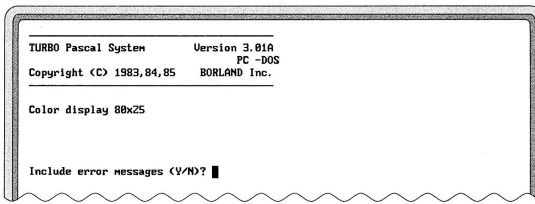
8.3 Turbo-Pascal Version 3

8.3.1 Aufruf

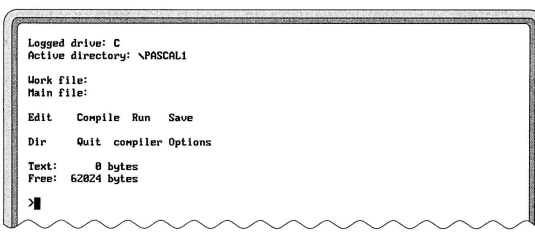
Das Turbo-Pascal-System wird aufgerufen, indem Sie

turbo [↵]

über die Tastatur eintasten. Auf dem Bildschirm erscheint die Einschaltmeldung:



In der Einschaltmeldung werden Sie gefragt, ob die Datei, in der die Fehlermeldungen erläutert werden, in den Arbeitsspeicher geladen werden soll. Beim Überprüfen von Programmen ist es sinnvoll, wenn die im Programm enthaltenen Fehler nicht nur durch eine Fehlernummer gekennzeichnet, sondern auch durch einen Text erklärt werden. Drücken Sie deshalb die Taste [Y] für yes.
Anschließend erscheint auf dem Bildschirm das Hauptmenü:

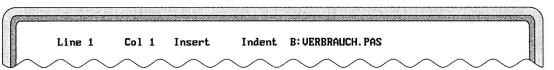


Die in dem Hauptmenü angezeigten Kommandos werden aufgerufen, indem der jeweils hervorgehobene Buchstabe eingegeben wird.

8.3.2 Ablauf einer Arbeitssitzung

Programmeingabe
Im ersten Schritt geben Sie an, in welchem Laufwerk und in welchem Verzeichnis die Pascal-Datei steht oder aufgebaut werden soll. Im

nächsten Schritt öffnen Sie durch [W] das Eingabefeld für die Angabe des Dateinamens. Da das System automatisch die Endung **.PAS** anhängt, brauchen Sie diese nicht anzugeben. Heißt die Datei beispielsweise VERSU1.PAS, so geben Sie lediglich VERSU1 ein.
Das System sucht die Datei auf dem angegebenen Laufwerk im angegebenen Verzeichnis. Existiert dort noch keine Datei, meldet das System **new file**. Mittels [E] schalten Sie dann in den Editorbildschirm. Hier geben Sie nun das Programm ein oder bearbeiten ein bereits bestehendes.
Am oberen Bildschirmrand ist die sogenannte Statuszeile dargestellt.



Sie zeigt an, daß sich der Bildpunkt (Cursor) in der Zeile (Line) 1, Spalte (Col) 1 befindet. Wenn Sie einmal die Leertaste drücken, so sehen Sie, daß hinter Col nunmehr 2 steht.
Insert bedeutet, daß sich der Editor im Einfügemodus befindet. Wenn Sie die Taste [Einfg] betätigen, ändert sich die Anzeige in **Overwrite**. Sie haben den Editor umgestellt in den Überschreibmodus. Was das zu bedeuten hat, wird weiter unten an einem Beispiel erläutert.

Hinweise für die Programmeingabe:
Bei der Eingabe des Programmtextes ist es gleichgültig, ob Sie alles groß oder klein schreiben.
Jede Programmzeile schließen Sie mit [↵] ab.
Nach der Eingabe eines Hochkommas ´ muß die Leertaste betätigt werden, damit es am Bildschirm angezeigt wird.

Die Programmbearbeitung beenden Sie mittels der Tastenkombination [Strg] [K] [D]. Das System schaltet, nachdem Sie eine beliebige Taste angeschlagen haben, in das Hauptmenü zurück.

Compilieren
Bevor das Programm, das sich im Editorbildschirm befindet, ausgeführt werden kann, muß es in die Maschinensprache übersetzt werden. Diese Aufgabe übernimmt der Compiler, den Sie im Haupt-

menü mittels [C] aktivieren. Weist Ihr Programm syntaktische Fehler auf, so bricht der Compiler den Übersetzungslauf ab. In einer Fehlermeldung wird erklärt, welcher Fehler festgestellt wurde. Sie müssen dann den Fehler beheben, mittels der Tastenkombination [Strg] [K] [D] das Editieren beenden und erneut den Übersetzungslauf starten.

Speichern
Ist das Programm fehlerfrei übersetzt worden, sollten Sie es unbedingt speichern, damit es nicht bei den weiteren Arbeitsgängen versehentlich verloren geht. Die Speicherfunktion rufen Sie im Hauptmenü mittels [S] auf. Das Programm wird dann in dem Laufwerk und in der Datei gespeichert, die Sie zu Beginn der Arbeitssitzung angegeben haben.

Programmstart
Durch [R] wird die Ausführung eines Pascal-Programms gestartet. Wenn der Ablauf des Programms nicht in Ordnung ist, liegt ein logischer Fehler vor.

Beenden
Durch Eingabe von [Q] im Hauptmenü beenden Sie die Arbeitssitzung.

8.3.3 Editierkommandos

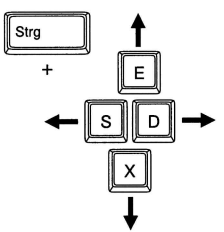
Der Editor bietet Ihnen eine Vielzahl von Funktionen an, die sehr nützlich sind bei der Eingabe und Bearbeitung von Programmen.

Cursorsteuerung
Die Cursorsteuerung kann entweder über die Pfeiltasten oder über eine Tastenkombination erreicht werden:

Funktion	Cursor-tasten	Tasten-kombination
Cursor		
ein Zeichen nach links	[←]	[Strg] [S]
ein Zeichen nach rechts	[→]	[Strg] [D]
ein Wort nach links	[Strg] [←]	[Strg] [A]

Funktion	Cursor-Tasten	Tasten-Kombination
ein Wort nach rechts	Strg →	Strg F
eine Zeile nach oben	↑	Strg E
eine Zeile nach unten	↓	Strg X
zum oberen Bildschirmrand	Strg Pos1	Strg Q E
zum unteren Bildschirmrand	Strg Ende	Strg Q X
zum linken Zeilenrand	Pos1	Strg Q S
zum rechten Zeilenrand	Ende	Strg Q D
auf das Textende	Strg Bild↓	Strg Q C
auf den Textanfang	Strg Bild↑	Strg Q R

Auch in Kombination mit der Strg-Taste kann der Cursor bewegt werden:



Verschieben von Texten auf dem Bildschirm

Funktion	Funktionstasten	Tastenkombination
Text eine Zeile nach unten rollen		Strg W
Text eine Zeile nach oben rollen		Strg Z
auf die nächste Seite blättern	Bild↓	Strg C
auf vorherige Seite blättern	Bild↑	Strg R

Löschen

Das Löschen kann zeichen-, wort- oder zeilenweise erfolgen.

Funktion	Tasten
Zeichen links vom Cursor löschen	←
Zeichen löschen, auf dem der Cursor steht	Entf oder Strg G
Wort rechts vom Cursor löschen	Strg T
Zeile, in der der Cursor steht, löschen	Strg Y
Rest einer Zeile, der rechts vom Cursor steht, löschen	Strg Q Y

Zeile einfügen

Um in einen bestehenden Text eine neue Zeile eingeben zu können, muß zuerst dafür Platz geschaffen werden. Sie fahren den Cursor auf die Zeile, vor der die neue Zeile eingefügt werden soll, und geben dann Strg N ein.

Eine andere Möglichkeit: Sie setzen den Cursor an das Ende der vorausgehenden Zeile und drücken die ↵-Taste. Das klappt jedoch nur, wenn der Einfügemodus eingeschaltet ist.

Blockoperationen

Für das Kopieren und Verschieben von Programmtexten benötigen Sie die Blockoperationen. Ein Block besteht aus mehreren aufeinanderfolgenden Zeichen oder Zeilen. Sie bilden den Block, indem Sie den Cursor auf das erste Zeichen stellen und die Tastenkombination Strg K B eintasten. Anschließend fahren Sie den Cursor auf das letzte Zeichen des gewünschten Blocks und geben Strg K K ein. Sie erkennen nun den Block, da er sich farblich von seiner Umgebung abhebt.

Anschließend können Sie folgende Operationen ausführen:

Funktion	Tastenkombination
Block löschen	Strg K Y
Block kopieren	Strg K C Der Block wird an der aktuellen Cursorposition eingefügt.
Block verschieben	Strg K V Der Block wird an die aktuelle Cursorposition verschoben.
Block speichern	Strg K W. Dadurch wird eine Eingabezeile geöffnet, in die Sie den Dateinamen eintragen. Geben Sie als Dateinamen lpt1: ein, so wird der Text ausgedruckt.
Datei einlesen	Strg K R + Dateiname
Blockmarkierung aufheben	Strg K H

8.3.4 Suchen und Ersetzen

Haben Sie eine Zeichenfolge im ganzen Text falsch geschrieben, so gibt es eine einfache Möglichkeit, den Fehler zu beheben.

Beispiel: Statt writeln wurde stets writlen geschrieben.

Mittels Strg Q A wird die Suche-Ersetze-Funktion eingeleitet. In dem Eingabefeld hinter *find:* tragen Sie die Zeichenfolge ein, die zu ersetzen ist. Die Eingabe schließen Sie mit ↵ ab. In der folgenden Eingabezeile hinter *replace with* tragen Sie die korrekte Zeichenfolge ein. Anschließend fragt das System mit der Meldung *options*, unter welchen Voraussetzungen ersetzt werden soll. Folgende Optionen können gegeben werden:

Option	Suche und ersetze ...
G	im gesamten Text ohne Rücksicht auf die aktuelle Cursorposition
B	rückwärts von der Cursorposition bis zum Textanfang
N	ohne weitere Nachfragen

Option	Suche und ersetze
	Wird N nicht angegeben, fragt das Programm jedesmal nach, ob das Ersetzen tatsächlich vorgenommen werden soll. Sie antworten dann entweder mit Y(es) oder N(o).
U	ohne einen Unterschied zwischen groß oder klein geschriebenen Wörtern
W	nur ganze Wörter Wird diese Option nicht gesetzt, so wird auch eine Zeichenfolge, die innerhalb eines Wortes steht und mit der Zeichenfolge des Suchwortes übereinstimmt, ersetzt. **Beispiel** renr (= Abkürzung für Rechnungsnummer) soll ersetzt werden durch rechnr. Ohne die Option W würde auch das Wort Tourenr geändert in Tourechnr.
i	ab der Cursorposition i-mal. i stellt eine Ganzzahl dar.

Innerhalb der Suche-und-Ersetze-Operation sind noch zwei weitere Kommandos von Bedeutung. Mittels [Strg] [L] wird der Suche-und-Ersetze-Vorgang fortgesetzt. Mit [Strg] [U] wird er dagegen abgebrochen.

8.3.5 Drucken eines Programms

Es gibt verschiedene Möglichkeiten, ein Turbo-Pascal-Programm auszudrucken. Eine Möglichkeit wurde bereits im Abschnitt Blockoperationen behandelt.
Ist ein Programm nicht länger als eine Bildschirmseite, können Sie mittels

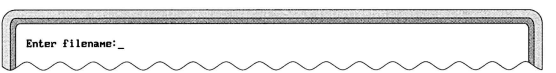
[⇧] [Druck]

den Bildschirminhalt über den Drucker ausgeben.
Schließlich können Sie auch ein Druckprogramm benutzen. Das Turbo-Pascal-System stellt Ihnen ein einfaches Druckprogramm unter dem Namen LISTER.PAS zur Verfügung. Mit LISTER.PAS können Sie Programme ausdrucken, die auf einer Diskette oder Festplatte gespeichert sind.

Vorgehensweise:
Schalten Sie den Drucker ein, und machen Sie ihn druckbereit. Das Druckprogramm LISTER.PAS wird aus dem Hauptmenü aufgerufen.

A:LISTER.PAS ↵

Starten Sie das Programm mit R. Am oberen Bildschirmrand erscheint die Meldung



Geben Sie nun den Namen der zu druckenden Programmdatei ein, zum Beispiel VERBRAUC.PAS ↵. Die Programmdatei wird jetzt gedruckt.

8.3.6 Zusammenfassung

Start des Turbo-Pascal-Systems und Aufruf des Editors

Gewünschte Aktion	Eingaben	Systemreaktion
Start des Turbo-Pascal-Systems	turbo ↵	
Fehlertextedatei einbinden	Y	Das Hauptmenü wird angezeigt.
Das Laufwerk A: als aktuelles Laufwerk einstellen	L A: ↵ Leertaste	Das Hauptmenü wird angezeigt.
Namen der Arbeitsdatei vereinbaren	W Eingabe des Dateinamens ↵ Leertaste	 Wenn Datei bereits auf der Diskette besteht, wird sie eingelesen. Das Hauptmenü wird angezeigt.

Gewünschte Aktion	Eingaben	Systemreaktion
Aufruf des Editors	E	Wenn Datei bereits besteht, wird sie auf dem Bildschirm angezeigt.
Beenden des Editierens	Strg K D Leertaste	Das Hauptmenü wird angezeigt.

Dateinamen

Ein Dateiname besteht aus 1 bis 8 Zeichen. Das erste Zeichen muß ein Buchstabe sein. Hinter dem Namen kann eine Dateikennung angegeben werden. Wird keine angegeben, hängt das System automatisch .PAS an.

Als Dateikennung dürfen die Namen .BAK, .CHN, .COM und .CMD nicht benutzt werden, da sie im Turbo-Pascal-System zu anderen Zwecken verwendet werden.

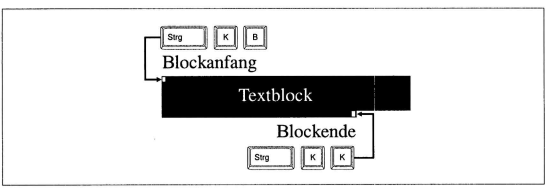
Wird eine Datei erneut abgespeichert, wird die bisherige .PAS-Version in .BAK umbenannt. Eine bereits bestehende .BAK-Version verschwindet, und die aktuelle Version wird unter .PAS gespeichert.

8.3.7 Übungen

1. Wozu wird der Editor benötigt?
   Mit dem Editor kann man Programmtexte...
2. Welche Aufgabe hat der Compiler?
   Der Compiler prüft das Quellprogramm auf... und erzeugt...
3. Welche Wirkung hat das Kommando Strg Y?
4. Auf der Arbeitsdiskette befinden sich die beiden Programmdateien TESTP.PAS und TESTP.BAK. Welche Datei enthält die aktuelle, letzte Version des Programms?
5. Aus wieviel Zeichen darf ein Dateiname höchstens bestehen?
6. Was bewirkt das Editorkommando Strg Q A?
7. Wie wird ein Block markiert?

8.3.8 Lösungen

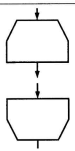
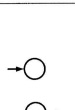
1. Mit dem Editor kann man Programmtexte eingeben und bearbeiten.
2. Der Compiler prüft das Quellprogramm auf syntaktische Richtigkeit und erzeugt das Maschinenprogramm.
3. Strg Y löscht die Zeile, in der der Cursor steht.
4. Die letzte Version des Programms steht in TESTP.PAS.
5. Der Dateiname darf aus höchstens 8 Zeichen bestehen.
6. Strg Q A leitet die Suche-und-Ersetze-Funktion ein.
7. Strg K B markiert den Blockanfang.
   Strg K K kennzeichnet das Blockende.



9 ANHANG

9.1 Symbole für den Programmablaufplan

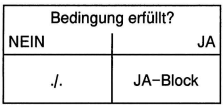
Sinnbild	Benennung	Einsatzmöglichkeiten
	allgemeine Verarbeitung (:=, Int, div) Ein- oder Ausgabe (read, readln) (write, writeln) (gotoxy, ClrScr) (Lst, ClrEol)	Zuweisung von Werten an Variablen und Konstanten, Berechnung von Ausdrükken, Stringmanipulationen. Angabe, um welche Ein- oder Ausgabe es sich handelt und welches Ein- oder Ausgabegerät angesprochen wird.
	Verzweigung (if... then) (if... then... else) (until) (case... of)	Je nachdem, ob eine oder mehrere Bedingungen zutreffen, verzweigt das Programm.(while) Bedingungsprüfung innerhalb von Wiederholungsschleifen. Verzweigungsliste mit mehreren Auswahlmöglichkeiten.
	Unterprogramm (procedure) (function)	Ein eigenständiger Programmblock wird vom übergeordneten Programm aus aufgerufen. Ersatzsymbol für einen eigenständigen Programmablaufplan.

Sinnbild	Benennung	Einsatzmöglichkeiten
	Wiederholungen: Zählergesteuert (for KV: = AW to EW do...) (for KV: = EW downto AW do...)	Anfangs- und Endebegrenzung einer Zählschleife. Kontrollvariable (KV) Anfangswert (AW) Endwert (EW)
	Verbindung (nächste Anweisung)	Vorzugsrichtlinien sind a) von oben nach unten b) von links nach rechts Ablauflinien mit Pfeilspitzen verdeutlichen die Richtung.
	Verbindungsstelle (label) unbedingter Sprung (goto)	Eine Verbindungsstelle darf als Eingangsstelle jeweils nur ein einziges Mal vorhanden sein. Als Ausgangsstelle darf die Verbindungsstelle mehrfach vorkommen.
	Grenzstelle (end)	Anfang und Ende eines Programms, einer Prozedur oder einer Funktion.
	Bemerkung	Das Sinnbild kann zu jedem anderen Sinnbild zur näheren Erläuterung hinzugefügt werden.

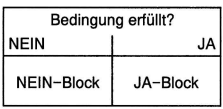
9.2 Symbole für das Struktogramm

Verarbeitungsschritt a
Verarbeitungsschritt b
Verarbeitungsschritt c

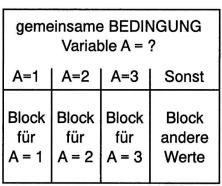
Sequenz oder Folge der Verarbeitungsschritte a, b und c



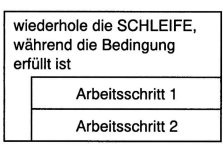
Zweiseitige Entscheidung mit bedingter Verarbeitung
Der Steuerungsteil (bei NEIN) ist leer und wird deshalb nicht ausgeführt.



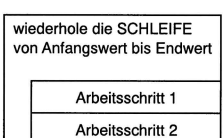
Zweiseitige Entscheidung mit einfacher Alternative
Entsprechend der im Kopfteil angegebenen Bedingung wird einer der beiden Blöcke abgearbeitet.



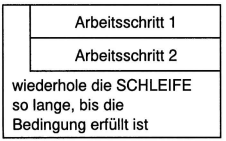
Mehrseitige Entscheidung mit mehrfacher Alternative
Nach der im Kopfteil angegebenen BEDINGUNG wird in den Block verzweigt, der mit dem Wert der Variablen übereinstimmt. Trifft kein Wert zu, so wird die Alternative «Sonst» ausgeführt.



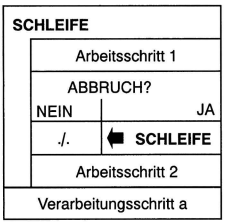
Wiederholung mit vorausgehender Bedingungsprüfung
Die Schleife wird so lange wiederholt, bis die Bedingung nicht mehr zutrifft.



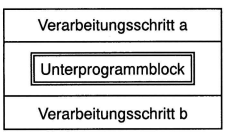
Wiederholung mit vorausgehender Bedingungsprüfung
Besonderheit: Zählschleife mit fest vorgegebener Anzahl von Durchläufen.



Wiederholung mit nachfolgender Bedingungsprüfung
Die Schleife wird so lange wiederholt, bis die Bedingung eintritt. In jedem Fall wird die Schleife einmal durchlaufen.



Abbruchanweisung innerhalb einer SCHLEIFE mit bedingter Verarbeitung
Wird im Verlauf des Programms die Abbruchanweisung erreicht, so wird entsprechend dem vermerkten Aussprunghinweis (hier SCHLEIFE) die Struktur verlassen.



Der Block als **Ersatzzeichen für ein anderes Struktogramm** ist zwar kein Programmkonstrukt im Sinne der DIN 66262, findet aber in der DIN 66261 Verwendung. Die hier gewählte Darstellung in der Form der Doppellinie soll den Ersatzcharakter für andere Struktogramme, wie Funktionen oder Prozeduren, verdeutlichen.

9.3 ASCII-Tabelle

ASCII-Code	unter Turbo PASCAL												(Industriestandard Set#2)							
NULL	☺	☻	♥	♦	♣	♠	BEL	BS	o	LF	♂	♀	CR	♫	☼	►	◄	↕	‼
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
¶	§	▬	↨	↑	↓	→	←	∟	↔	▲	▼	BLANK	!	"	#	$	%	&	'
20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
(	)	*	+	,	−	.	/	0	1	2	3	4	5	6	7	8	9	:	;
40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
<	=	>	?	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	`	a	b	c
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99
d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w
100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119
x	y	z	{	|	}	~	⌂	Ç	ü	é	â	ä	à	å	ç	ê	ë	è	ï
120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139
î	ì	Ä	Å	É	æ	Æ	ô	ö	ò	û	ù	ÿ	Ö	Ü	¢	£	¥	₧	ƒ
140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
á	í	ó	ú	ñ	Ñ	ª	º	¿	⌐	¬	½	¼	¡	«	»	▒	▓	▓	|
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179
┤	╡	╢	╖	╕	╣	║	╗	╝	╜	╛	┐	└	┴	┬	├	─	┼	╞	╟
180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199
╚	╔	╩	╦	╠	═	╬	╧	╨	╤	╥	╙	╘	╒	╓	╫	╪	┘	┌	█
200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219
▄	▌	▐	▀	α	ß	Γ	π	Σ	σ	µ	τ	Φ	Θ	Ω	δ	∞	ø	ε	∩
220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
≡	±	≥	≤	⌠	⌡	÷	≈	°	·	·	√	ⁿ	²	■					
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255				

9.4 Literaturverzeichnis

Borland GmbH (Hrsg.): Borland Pascal mit Objekten 7.0
– Benutzerhandbuch
– Referenzhandbuch
– Programmierhandbuch
Langen 1992
Gottfried, Byron S.: Programmieren mit Pascal, Hamburg New York 1986
Herschel, Rudolf: Turbo Pascal, München Wien 1985
Kuhlmann, Gregor: Informatik – Finanzmathematik, Bad Homburg vor der Höhe 1984
Kuhlmann, Gregor: Quickmanual Turbo Pascal, Rinteln 1987
Renner, Gerhard: Turbo Pascal, Würzburg 1986

Hinweis

Autor, Herausgeber und Verlag haben sämtliche Angaben, Hinweise und Beispiele, die in diesem Buch aufgeführt sind, sorgfältig geprüft. Dennoch können Fehler nicht völlig ausgeschlossen werden. Autor, Herausgeber und Verlag können deshalb keine Gewährleistung für die einwandfreie Funktion aller Angaben, Hinweise und Beispiele übernehmen. Für etwaige Folgeschäden an Geräten und Programmen, die durch die Benutzung der Inhalte dieses Buchs entstehen können, wird keine Haftung übernommen.
Für Hinweise auf Fehler und Verbesserungsvorschläge sind wir sehr dankbar.

Produkt- und Warennamen

IBM ist ein eingetragenes Warenzeichen von International Business Machines Corporation.
MS-DOS ist ein eingetragenes Warenzeichen von Microsoft Corporation.
Turbo-Pascal ist eingetragenes Warenzeichen von Borland Inc.

9.5 Sachwortregister

$I 290
$V 245
.BAK 36, 295
.PAS 26

abs 262
abzählbarer Datentyp 63
Aktionsschalter 19
aktueller Parameter 224
Algorithmus 81
AND 172
arctan 265
Ausdruck überwachen 93
Ausgabebildschirm 21
Ausgabedaten 78
Ausgabefenster 20
Ausgabeformatierung 68
Auswahl 107f
Auswahl, einseitig 107f
Auswahl, mehrseitig 137f
Auswahl, zweiseitig 121f
Auswahlmarke 140

Bedingungsausdruck 113, 172
Betriebssystem aufrufen 26
Block aus Datei lesen 48
Block in Datei schreiben 48
Block kopieren 45
Block löschen 44
Block markieren 42
Blockoperation 328
boolean 63
break 170
byte 72

case 137
char 62
chr 267
Clipboard 303
clreol 174, 268
clrscr 61, 268
comp 72
compile 306
Compiler 33, 283
Compiler-Option 243, 310f
compilieren 325
concat 271
const 84
copy 271, 303
cos 265
crt 61
crtexit 268
crtinit 268
Cursorsteuerung 326
cut 303

Dateiname 35
Datenanalyse 78
Datentypen 62
Debug 283f
Debugger 292
dec 266
delay 176, 178, 272
delete 269
delline 213, 268
Destination 307
Dialogfenster 277
Diskriminante 125
div 70
Dokumentation 99f
double 72
Drucken 330f
Druckerinstallation 278
Druckmaske 68

Editfenster 18, 21
Einfügemodus 41
Eingabedatei 78
Eingabekorrekturen 65
environment 315
error 272
Erstlesen 177
exit 272
exp 263
extended 72

false 72, 114
Fenster aktivieren 21
Fenster aufheben 23
Fenster installieren 30
Fenster verschieben 22
Folgelesen 177
for – downto – do 149, 167
for – to – do 149, 161
formaler Parameter 224
Forward-Referenz 245
frac 262
Funktion 207, 250

getdate 273
Gleitkomma 67 f
Gleitkommaberechnung 291
globale Variable 214
goto 168
gotoxy 174, 269

halt 273
Haltepunkt 284, 308
Hilfetext 25

if 113
if-then 107, 110
if – then – else 121 ff
inc 266
insert 269
insline 269
int 263
integer 62, 72
ioresult 273, 290, 312
Iteration 148

keypressed 273
Kommentar 132
Konfigurationsdatei 294
Konstantenvereinbarung 84
Kontrollvariable 165
Kopier- und Löschfunktion 303

label 169
Längenbyte 203
Laufvariable 165
Laufzeitfehler 98, 282, 289
Leeranweisung 130
length 271
Listensymbol 278
ln 264
Logiktest 88
lokale Variable 214
longint 72
lowvideo 269
lst 196

Markierungsfeld 278
Maus 22
Maus-Option 296
maxint 72
Menüauswahl 17
Menüleiste 16
mod 70
Modul 240
modularisierte Programmierung 240
mouse 317

Name 59
normvideo 269
NOT 173

odd 266
Oder-Verknüpfung 173
Öffnen 279
Open 302
Option 288
OR 173
ord 267
ordinaler Datentyp 63

Parameterliste 225
paste 303
persistente Blöcke 295
pi 72
Plausibilitätskontrolle 171
pos 271
pred 266
printer 197
Problemanalyse 77
procedure 209
program 59
Programm ausführen 36
Programm eingeben 29, 85
Programm laden 31
Programm speichern 34
Programmablaufplan 81
Prozedur 207
Prozedur, eingeschachtelt 214
Prozedur, mit Variablenübergabe 232
Prozedur, mit Wertübergabe 222
Prozedur, parameterlos 209
Pseudocode 81

quadratische Gleichung 125
Quellprogramm 33
Quellprogramm drucken 37

random 178, 274
randomize 273
readkey 273
readln 65
real 62, 72
reguläre Ausdrücke 51
Rekursion 259
repeat – until 149, 150
round 268

save 302
save all 302
Schachtelung 124
Schaltfeld 278
Schleife 149
Schließsymbol 24
search 304
Selektor 140
shortint 72
sin 265
single 72
Speicher unter ... 278
sqr 113, 265
sqrt 128, 265
Standardprozedur 71
Start 285
str 270
String 62 f, 196 f
Struktogramm 81
succ 266
Suchen 281
Suchen und Ersetzen 49, 329
Syntaxfehler 87
Syntax-Option 291
Syntaxprüfung 33, 86

Tabelle 186
Textkonstante 66
Tools 287, 293
Trace 90 ff, 112
Trace abbrechen 95
true 72, 114
trunc 268
Type-Vereinbarung 228

Überschreibmodus 41
Übersetzen 33
Überwachungsfenster 21
Umgebung 294
units 73
Unterbereiche 144
Unterverzeichnis festlegen 26 ff
upcase 274
uses 61

val 270
var 61 f, 236
Variable inspizieren 93, 307
Variable modifizieren 307
Variablenparameter 235
Verbundanweisung 115
Verfeinerung, schrittweise 126
Verzeichnisse 295
Vorgaben 294

Wahrheitswert 63
Watchfenster 20
Wertbelegungstabelle 108
Werteparameter 225
Wertfortschreibung 190
Wertzuweisung 69
wherex 269
wherey 269
while – do 149, 156
word 72
write 66
writeln 66

Zählervariable 165
Zeichenkette 196 f
Zeile einfügen 328
Zeile löschen 328
Zoomen 22
Zwischenablage 280

Grundkurs Computerpraxis

Grundkurs Computerpraxis bietet Schulungskurse im Taschenbuchformat, die systematisch die Grundlagen für gängige Anwendungsprogramme vermitteln. Von erfahrenen Didaktikern in der Ausbildungspraxis erprobt, bieten sie einen effektiven Zugang zu Standardsoftware, Betriebssystemen und Programmiersprachen.

Benno Brudermanns / Gregor Kuhlmann / Lutz Seemann
Office 97: Lösungen mit Access
(rororo computer 19870)
Access für Windows 95
(rororo computer 19832)

Reiner Backer
Assembler
(rororo computer 19249)

Helmut Erlenkötter / Volker Reher
Programmiersprache C *Visual C, Quick C, MS-C/C++, Turbo-, Borland C++*
(rororo computer 18166)
C++ für Windows 95 / NT *32-Bit-Programmierung mit MFC*
(rororo computer 19866)
Objektorientiertes Programmieren in C++
(rororo computer 19282)

Gregor Kuhlmann / Alexander Parkmann / Joachim Röhl / Johannes Verhuven
Computerwissen *für effiziente Nutzer*
(rororo computer 19896)

rororo computer

Helmut Erlenkötter / Volker Reher
Delphi für Windows
(rororo computer 19858)

Paul Klimsa
Desktop Video *Videos digital bearbeiten (mit CD-ROM)*
(rororo computer 19885)

rororo computer wird herausgegeben vom *Ludwig Moos*. Ein Gesamtverzeichnis aller lieferbaren Titel finden Sie in der *Rowohlt Revue*. Vierteljährlich neu. Kostenlos in Ihrer Buchhandlung.

Rowohlt im Internet:
www.rowohlt.de

Grundkurs Computerpraxis

Gregor Kuhlmann / Friedrich Müllmerstadt
SQL *für DOS, Windows, OS/2 und Unix*
(rororo computer 19289)

Gregor Kuhlmann
Turbo-Pascal *bis Version 7.0*
(rororo computer 18148)
Turbo-Pascal für Fortgeschrittene
bis Version 7.0
(rororo computer 18155)

Gregor Kuhlmann / Ulrich Bornschein / Lutz Seemann
Turbo-Pascal für Windows
Objektorientierte Programmierung und Grafik (bis Version 7.0)
(rororo computer 19235)

Hans-Josef Heck
UNIX
(rororo computer 18167)

Helmut Erlenkötter
Visual Basic 5
(rororo computer 19820)

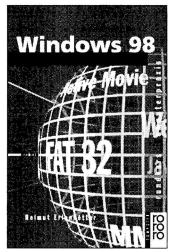
Helmut Erlenkötter / Volker Reher
Windows 95
(rororo computer 19801)

Windows 3.1 *bis Version 3.1*
(rororo computer 19230)

Windows NT *Version 4*
(rororo computer 19864)

Helmut Erlenkötter
Windows 98
(rororo computer 19893)

rororo computer



Joachim Röhl / Johannes Verhuven
Office 97: Lösungen mit Word
(rororo computer 19867)
Word für Windows 95
(rororo computer 19830)
Word 6 für Windows
(rororo computer 19286)

rororo computer wird herausgegeben von *Ludwig Moos*. Ein Gesamtverzeichnis aller lieferbaren Titel finden Sie in der *Rowohlt Revue*. Vierteljährlich neu. Kostenlos in Ihrer Buchhandlung.

Rowohlt im Internet:
www.rowohlt.de

Visual QuickSteps

Viele Lösungswege verlaufen im Sande, weil sie zu umständlich und zu abstrakt sind. **Visual QuickSteps** weisen den direkten Weg: anschaulich, intuitiv, ergebnisorientiert. Diese Computertaschenbücher zeigen Ihnen jeden einzelnen Arbeitsschritt als Bildschirmdarstellung. Damit Sie immer schon vorher wissen, was nachher passiert.

Excel für Windows 95
in 700 Bildschritten
(rororo computer 19840)

FrontPage
in 500 Bildschritten
(rororo computer 19899)

HTML für das Web
in 500 Bildschritten
(rororo computer 19894)

Windows 98
in 500 Bildschritten
(rororo computer 60060)

Windows 95
in 500 Bildschritten
(rororo computer 19813)

rororo computer

Word 6 für Windows
in 800 Bildschritten
(rororo computer 19276)

Word für Windows 95
in 450 Bildschritten
(rororo computer 19839)

rororo computer wird herausgegeben von Ludwig Moos. Ein Gesamtverzeichnis aller lieferbaren Titel finden Sie in der *Rowohlt Revue*. Vierteljährlich neu. Kostenlos in Ihrer Buchhandlung.

Rowohlt im Internet:
www.rowohlt.de

3432/8

Keine Panik!

Keine Panik! hilft mit eingängigen Anleitungen und griffigen Tips auch Computernovizen, die Computerwelt zu verstehen und die tägliche Arbeit sicher zu bewältigen. Für die Orientierung im Dschungel der Fachbegriffe sorgen die beiden Nachschlagewerke von Hans Herbert Schulze.

Keine Panik!:

Hans Ollig
Daten sichern, Daten schützen
(rororo computer 19879)

Excel für Windows 95 *Sicher nutzen*
(rororo computer 19837)

Ingo Steinhaus
Internet *Sicher nutzen*
(rororo computer 19861)
Windows 98 *Sicher nutzen*
(rororo computer 60061)
Windows 95 *Sicher nutzen*
(rororo computer 19821)
Word für Windows 95 *Sicher nutzen*
(rororo computer 19836)

Ingo Steinhaus / Hans Ollig
Personal Computer *Sicher nutzen*
(rororo computer 19838)

rororo computer

Nachschlagewerke:

Hans Herbert Schulze
PC-Lexikon *Fachbegriffe schlüssig erklärt*
(rororo computer 19828)

Computer-Englisch
(rororo computer 19871)

Computerkürzel *Lexikon der Akronyme, Kurzbefehle und Abkürzungen*
(rororo computer 19888)

rororo computer wird herausgegeben von Ludwig Moos. Ein Gesamtverzeichnis aller lieferbaren Titel finden Sie in der *Rowohlt Revue*. Vierteljährlich neu. Kostenlos in Ihrer Buchhandlung.

Rowohlt im Internet:
www.rowohlt.de

Online

Experten sind sich einig: Die Welt geht online. Die Telekommunikation, schon heute aus dem beruflichen und privaten Leben nicht mehr wegzudenken, gilt als eine Schlüsseltechnologie der Zukunft. Ob Telebanking oder computergestütztes Verkaufen, Austausch über Mailboxen oder das Nutzen des World Wide Web im Internet – mit den Online-Büchern bei *rororo computer* gehen Sie auf spannende und unterhaltsame Datenreisen.

Uwe Kreisel /
Pamela Ann Tabbert
E-mail English *Fit in Kommunikation und Technik*
(rororo computer 19876)

Phyllis Davis /
Deborah Craig
FrontPage *in 800 Bildschritten*
(rororo computer 19899)

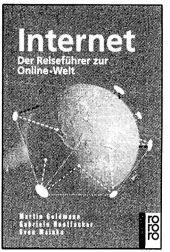
Elisabeth Castro
HTML für das Web *in 500 Bildschritten*
(rororo computer 19894)

Martin Goldmann /
Gabriele Hooffacker /
Sven Mainka
Internet *Der Reiseführer zur Online-Welt*
(rororo computer 19877)

Ingo Steinhaus
Keine Panik! Internet *Sicher nutzen*
(rororo computer 19861)

Gabriele Hooffacker /
Rena Tangens
Online-Guide Frauen & Netze
(rororo computer 19873)

rororo computer



Gabriele Hooffacker /
Peter Lokk
Online-Guide Politik & Gesellschaft
(rororo computer 19863)

Ernst Tiemeyer
Online mit Office *Web-Piblikationen und Datenaustausch im Internet / Intranet*
(rororo computer 19890)

Ingo Steinhaus
Online recherchieren *Ökonomische Wege zu Informationen*
(rororo computer 19875)

rororo computer wird herausgegeben von *Ludwig Moos*. Ein Gesamtverzeichnis aller lieferbaren Titel finden Sie in der *Rowohlt Revue*. Vierteljährlich neu. Kostenlos in Ihrer Buchhandlung.

Rowohlt im Internet:
www.rowohlt.de

Grafik & Design Werkstatt

Die Reihe **Grafik & Design Werkstatt** liefert das Know-how für die optimale Umsetzung von Gestaltungsideen am Apple Macintosh und am PC.

Cyrus Dominik Khazaeli
Crashkurs Typo und Layout
Vom Zeilenfall zum Screendesign
(rororo computer 19815)

Cyrus Dominik Khazaeli
Multimedia mit Director *Projektplanung und Interfacedesign (mit CD-Rom, für Mac und PC)*
(rororo computer 19848)

Gabriele und Jochen Trumpetter
Illustrator *Lösungen für Anwender (bis Version 6)*
(rororo computer 19865)

Kathrin Douté
FreeHand *Lösungen für Anwender (bis Version 5)*
(rororo computer 19853)

Kathrin Douté /
Andreas Pankalla
FreeHand 7 + 8 *Lösungen für Anwender (für Mac und PC)*
(rororo computer 19892)

Photoshop 4 *Lösungen für Anwender (mit CD-Rom, für Mac und PC)*
(rororo computer 19889)

rororo computer

Kathrin Douté
Quark XPress *Lösungen für Anwender (bisVersion 3)*
(rororo computer 19262)

Kathrin Douté
QuarkXPress 4 *Lösungen für Anwender (für Mac und PC)*
(rororo computer 19860)

rororo computer wird herausgegeben von Ludwig Moos. Ein Gesamtverzeichnis aller lieferbaren Titel finden Sie in der *Rowohlt Revue*. Jedes Vierteljahr neu. Kostenlos in Ihrer Buchhandlung.

Rowohlt im Internet:
www.rowohlt.de

Die Programmiersprache Turbo-Pascal, die unter allen wichtigen Betriebssystemen von PCs läuft, hat sich in wenigen Jahren durchgesetzt. Sie wird von Programmieranfängern wie professionellen Programmierern gleichermaßen geschätzt. Ihr komfortables Textprogramm erleichtert die Eingabe und Bearbeitung von Programmen, die Menü-Steuerung ermöglicht eine einfache Handhabung. Mit dieser strukturierten Einführung, die praxisnah, anschaulich und Schritt für Schritt vorgeht, läßt sich ohne große Vorkenntnisse lernen, wie Programme unter Turbo-Pascal entwickelt und zum Laufen gebracht werden.

BIS VERSION 7.0

ISBN 3-499-18148-7

9 783499 181481

DM 19,90

ÖS 147,00

rororo COMPUTER